Frank Steinicke

Multimodal Metaphors for Generic Interaction Tasks in Virtual Environments

Münster
- 2006 -

Fach Informatik

Multimodal Metaphors for Generic Interaction Tasks in Virtual Environments

Inaugural-Dissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften im Fachbereich
Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von

Frank Steinicke
aus Rheine

- 2006 -

Dekan:	Prof. Dr. Klaus H. Hinrichs
Erster Gutachter:	Prof. Dr. Klaus H. Hinrichs
Zweiter Gutachter:	Prof. Dr. Antonio Krüger
Tag der mündlichen Prüfung:	29.06.2006
Tag der Promotion:	29.06.2006

Contents

List of Figures

List of Tables

Listings

Abstract

Virtual reality (VR) systems utilize additional input and output channels in order to make interactions in virtual environments (VEs) more intuitive. Furthermore, by exploiting these additional channels the user's immersion into the virtual world is increased. VR technologies have the potential to provide a better insight into complex datasets, although high demands on the ability of the user to interact in the virtual space are made. When interacting in VR-based applications the cognitive effort for accomplishing interaction tasks is higher compared to the effort required when interacting via two-dimensional user interfaces. This is due to the fact that the manifold interaction options provided by VR systems exceed the possibilities of the real world such that users may be overstrained. For example, in VR-based applications users can control additional degrees of freedom in order to accomplish complex interaction tasks such as virtual flying, grabbing of distant objects or performing arbitrary manipulations of virtual objects. The objective of *interaction metaphors* is to support users during an interaction process by reducing the cognitive effort required for performing certain interaction tasks by depicting complex tasks by commonly used techniques users are accustomed to. Furthermore, in order to increase the user's immersion into the virtual scene, these metaphors can address multiple sensorial channels, e.g., to give the user additional feedback and information during the interaction.

The contributions of this thesis include the design and evaluation of novel multimodal interaction metaphors for generic interaction tasks in VEs. To enhance the design process of these metaphors, the generic VR software system VR^2S has been developed. The proposed interaction metaphors provide users of VR-based applications with intuitive interaction concepts, which improve efficiency when performing certain interaction tasks, e.g., selection and manipulation of virtual objects. All developed metaphors are based on VR^{2}S including the *improved virtual pointer metaphor* that advances selection and manipulation of virtual objects, the *dual-purpose interaction metaphor*, which allows to use desktop-based interaction paradigms in VR, and *VR widgets* affording the usage of three-dimensional virtual menus in VR-based applications. Moreover, approaches for an advanced exploration of virtual worlds are presented. First, strategies to facilitate *collaborative interaction* in projection-based VR systems are proposed. Second, concepts for the simulation of *global illumination phenomena* in VEs are introduced in order to enhance seamlessly merging of real-world and virtual objects. All introduced concepts have been evaluated in different scenarios. These case studies have shown the usefulness of the proposed interaction strategies in interactive geovisualization as well as seismic volume visualization environments. The interaction concepts proposed in this thesis have been tested for both application areas in two prototype VR system environments which are described in more detail.

Acknowledgements

This dissertation is the result of my research work at the Department of Computer Science at the Westfälische Wilhelms-Universität Münster.

I am very grateful to my adviser Prof. Dr. Klaus H. Hinrichs for giving me this opportunity and his constant support. He introduced me to computer graphics and provided me with optimal working conditions. With many helpful comments and his enthusiasm for research and writing, he gave continuous stimuli for my own work.

My thanks go to Prof. Dr. Antonio Krüger for his readiness to be the co-referee of this thesis and for his valuable comments and support during several projects in which I have been involved.

I am very grateful to Dr. Timo Ropinski and Jennis Meyer-Spradow, who came up with helpful comments after reading a preliminary version of this thesis. Moreover they deserve thanks for many fruitful discussions and for the fun that we have.

And I am also very grateful to members and alumni of our working group who have always been good colleagues and maintained the excellent working atmosphere, Evelyn Egelkamp, Dr. Ludger Becker, Henrik Blunck, Michael Jacob, Dr. Christian Kemmer, Jennis Meyer-Spradow, Dr. Timo Ropinski and PD Dr. Jan Vahrenhold.

I thank the students and graduates of our department who have implemented parts of VR^{2}S and the applications described in this thesis, and who have participated in usability studies.

Next, I am very thankful to my parents Alwine and Johann Steinicke, and my brother Marc for their support and encouragement. Furthermore Marc deserves thanks for preparing professional animations and video material for several presentations.

Last but certainly not least, I am extremely grateful to Bianca Grommel for encouraging me to start my research work and for her loving support.

Chapter 1

Introduction

The evolution of novel user interface technologies further improves concepts and strategies for the interaction between users and computer systems. Human-computer interaction (HCI) evolved into a research discipline that addresses the design and evaluation of interactive user interface systems in order to make interaction with computers more intuitive. Desktop-based interaction is often insufficient in cases where intuitive and natural interfaces are desired, for example, when interacting with complex three-dimensional datasets. For such purposes sophisticated virtual reality (VR) technologies have been developed in order to provide advanced interfaces that offer great potential for HCI.

The research field of VR has been initiated in the 1960s when Ivan Sutherland created the first interactive system involving a tracked head-mounted display (HMD) and real-time three-dimensional computer graphics ([Sut68]). In comparison to currently available technology the system was crude in terms of computing and rendering power. However, all basic components that make up today's VR systems were present in this prototype.

From there on, there has been continual research in the area of real-time computer graphics and interactive virtual environments (VEs). Hardware technology that facilitates to render complex 3D scenes at interactive frame rates is permanently refined, e.g., thousands of different textured objects can be displayed even on large screens sufficiently fast. Tracking technologies provide the possibility to receive and process three-dimensional data for several tracked objects or devices in real-time. Moreover, VR input and output devices can address multiple human sensory modalities enabling users to interact with virtual worlds in an advanced manner. For example, auditory feedback creates the illusion of audio sources propagating from certain locations in the 3D world, and haptic devices allow users to touch and feel virtual objects; there are even approaches using olfactory and gustatory input and output in virtual reality ([YNTT03]).

Besides improvements regarding hardware technology, also software enhancements have been made; computer graphics algorithms allow rendering a virtual scene with sufficient visual quality at interactive frame rates. Furthermore, software systems and toolkits have been introduced to aid the rapid development of interactive applications. In particular, VR software systems ease the development of VR-based applications by handling rendering issues and by providing interfaces to tracking systems as well as input and output devices. Hence, developers can focus on the design of the virtual environment and the interaction processes in the VE rather than technical issues, e.g., implementing certain graphics effects.

VR technologies, in particular stereoscopic displays as well as sophisticated input and output devices, have the potential to provide a better insight into complex datasets. However, interaction in VR-based environments makes high demands on the ability of the user to interact in a three-dimensional virtual space. This is due to the fact that in contrast to two-dimensional user interfaces the user can specify position and orientation simultaneously via the pose of arbitrary 3D input devices. Moreover, the user can modify virtual objects without any restrictions in terms of the manipulability of the objects. Thus, when accomplishing certain interaction tasks in VR-based applications the cognitive effort for a user is higher compared to the effort required to interact via two-dimensional user interfaces.

The goal of this thesis is to improve the interaction between the user and a VR system by providing novel multimodal interaction metaphors. These metaphors aim to ease interaction processes by depicting complex procedures with commonly used approaches and techniques, and addressing multiple human sensorial channels. In this context it is important to understand how humans interact with computers and, in particular, how humans perceive information with the different sensorial modalities. Hence, in this thesis the main modalities, i.e., the visual, auditory and haptic systems, are considered.

The development, implementation and evaluation of the proposed interaction metaphors are based on a generic VR software system described in this thesis. Furthermore, case studies showing the usefulness of the proposed interaction concepts in interactive geovisualization as well as seismic visualization applications are discussed, and example VR system setups are described for which the proposed concepts have been evaluated in usability studies.

The remainder of this thesis is structured as follows. In Chapter 2 fundamental concepts of HCI are discussed. In particular, the main factors contributing to an interactive system are analyzed in detail. To obtain a basic overview of the capabilities and limitations that affect the user's ability to interact with a computer system, human factors are considered. Furthermore, different computer systems are described, with special consideration of the corresponding input and output devices in terms of their influence on the interaction process. Afterwards, the dialog between user and system is discussed in order to describe different models of interaction.

In Chapter 3 the generic VR software system VR^2S is presented, which allows the rapid development of multimodal interaction strategies for VR-based applications. The main components of a VR system are identified, and pipeline concepts for rendering visual, auditory and haptic effects are explained. Finally, the architecture and benefits of VR^{2}S are described by means of the integration and usage of basic components that constitute VR systems.

Chapter 4 discusses direct interaction metaphors for object selection and manipulation. The *improved virtual pointer metaphor* is introduced and its integration into VR^{2}S is described. Furthermore, a *dual-purpose metaphor* for both VR- as well as desktop-based interaction is proposed, and the integration of *VR widgets* into VR^{2}S to support VR-based system control is discussed. Usability studies evaluate the benefits and potentials of the proposed metaphors.

Chapter 5 presents two approaches, which further enhance VR-based exploration by focusing on *co-located interaction concepts* for large screen displays with the objective to enable groups of users to explore data in VR with confined resources. Furthermore, concepts for the *simulation of global illumination phenomena* in VEs are discussed, which allow to seamlessly merge real-world and virtual objects.

In Chapter 6 the technical characteristics of two example VR system setups, which provide users with semi-immersive VR, are described in detail. Moreover, case studies using the proposed interaction strategies for interactive geovisualization as well as seismic volume visualization applications are discussed.

Finally, Chapter 7 concludes this thesis and gives an overview of future research directions.

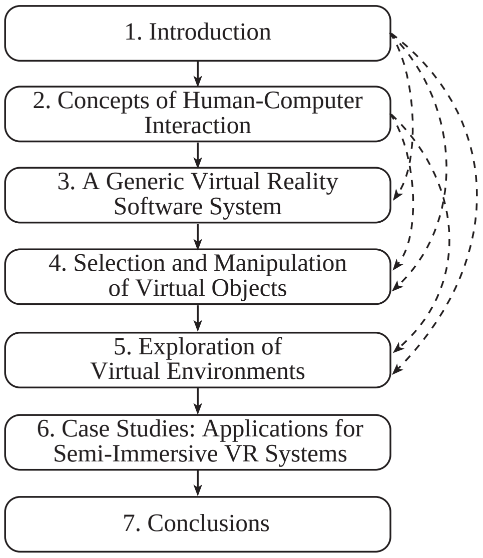

1. Introduction
2. Concepts of Human-Computer Interaction
3. A Generic Virtual Reality Software System
4. Selection and Manipulation of Virtual Objects
5. Exploration of Virtual Environments
6. Case Studies: Applications for Semi-Immersive VR Systems
7. Conclusions

Figure 1.1: Thesis road map and possible shortcuts.

When reading this thesis different approaches are possible. Figure 1.1 illustrates the "road map" to this thesis. Depending on the reader's previous knowledge and special interests different paths depicted in Figure 1.1 may be chosen while reading. For example, readers already having fundamental knowledge of HCI may skip Chapter 2 and proceed with Chapter 3, which presents VR^{2}S, or with Chapter 4, which introduces VR-based interaction metaphors.

Chapter 2

Concepts of Human-Computer Interaction

This chapter gives an introductory overview covering fundamental concepts of the interaction process between human users and computer systems. Since one focus of this thesis is on the development of interaction strategies for generic interaction tasks in virtual environments, related concepts regarding the interaction process between humans and computers are described in Section 2.1. All important factors contributing to an interactive system - the human user, the computer system and the interaction itself - are analyzed. To obtain a basic overview about the capabilities and limitations that affect the user's ability to interact with an arbitrary system, Section 2.2 discusses human factors with respect to perceptual and cognitive psychological aspects. Furthermore, different computer systems with special consideration of the corresponding input and output devices and their influence on the interaction process are described in Section 2.3. Finally, Section 2.4 focusses on the dialog between human users and computer systems in order to describe different models of interaction.

More detailed descriptions of interactive systems involving human users and computer systems can be found in [Dah06, Car01, DFAB98, HBC+92].

2.1 Human-Computer Interaction

The term *human-computer interaction (HCI)* was adopted in the beginning of the 1980s in order to describe the field of *user interface (UI)* systems, but originally this research field had a much broader scope than the design of user interfaces. Currently, there is no common agreement or general theory of the range of topics that form the area of human-computer interaction. Though, *ACM's Special Interest Group on Computer-Human Interaction* describes HCI as follows ([HBC+92]):

> *"Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the*

study of major phenomena surrounding them."

According to this definition, HCI considers all aspects that relate to the interaction between users and computers, i.e., design, implementation and evaluation of interactive computer systems in the context of certain tasks users want to accomplish. HCI is not restricted to a single user, who interacts with a single desktop computer. *User* either refers to an individual user, or a group of users working simultaneously or sequentially, co-located or distributed, while the term *computer* represents technology ranging from a single desktop-computer to embedded systems or ubiquitous computing approaches. *Interaction* denotes the communication between the user and the computer system.

During the past two decades, HCI evolved as a field of research concerned with both computer science research as well as applied social and behavioral science. In this context the study and practice of *usability* becomes a topic of major relevance ([RBML04, Car01]). In order to be established interactive computer systems and user interfaces have to be designed to meet certain requirements. First of all, these systems have to be *useful*, i.e., the system supports users to accomplish desired tasks. Moreover, interactive systems have to be *usable*, i.e., users are supported in performing tasks easily and intuitively. Another important issue is about the *experience* of using a new user interface; it should motivate users to make use of the user interface and provide them with enjoyment and satisfaction when using it. Thus, the objective of HCI is to improve the safety, utility, effectiveness, efficiency, and usability of interactive computer systems and to ensure that these systems integrate appropriately in the settings in which they are used.

In the 1970s, behavioral approaches to understand the usage of interactive computer systems developed rapidly, focusing on what users experience and how they interact with computer systems. These issues have initiated many industrial and research institutes to concentrate on the study of human factors and usability of software as well as hardware. This so-called *user-centric* development of interactive systems, in particular of user interfaces, involves different fields of research with different backgrounds, i.e., knowledge and expertise. Sciences that make a major contribution to HCI are

- *computer science*, which is engaged in engineering of user interfaces and appropriate software solutions,
- *cognitive psychology*, which investigates the application of theories of cognitive processes and analyzes user behavior,
- *sociology* and *anthropology*, which investigate interworking between technologies and users, in particular how humans and technical systems mutually adapt to each other, and
- *industrial design*, which deals with the design and development of products for interactive computer systems.

But also other disciplines such as *artificial intelligence*, *linguistics*, *philosophy* etc. are involved ([Car01, DFAB98]). Most of these disciplines include the development and empirical validation of models and theories as well as laboratory and field studies about users learning and using particular interactive computer systems. The combination of these different approaches became known as *usability engineering.*

Both the computer with its associated input and output devices as well as the capabilities and limitations of the human user using these technologies influence the nature of interaction and the design of the interface. Hence, in order to understand how humans interact with computers, both actors in such an interaction process are considered in the next sections.

2.2 Human Factors

For many domains the simulation of real-world behavior is an important prerequisite to enable comprehension, although *virtual environments* (VE), i.e., synthetically generated scenes, are not restricted to represent the real world. Hence, to enable innovative and advanced interaction concepts VEs often extend real-world appearance and behavior. In order to provide usable interfaces for the interaction between humans and computers, it is essential to understand human capabilities and limitations. For this purpose, sufficient knowledge of both *cognitive psychology* and in particular *perceptional* aspects are required ([BKLP01]). The latter includes knowledge about the way humans perceive information, in particular how information is perceived with the senses, whereas cognitive psychology focusses on mental aspects, how humans process information, in particular how humans learn, reason, or remember. Both aspects are briefly discussed in the following subsections

2.2.1 Multimodality

Humans process information by using different input and output channels simultaneously. *Perception* is understood as the process of acquiring, interpreting and organizing sensory information. *Senses* concern about the physiological methods for perception. In order to be accepted by the user a virtual environment has to address senses by providing the most important stimuli at their required qualities ([HL95]).

Therefore considering the amount of contribution of a particular sense gives valuable hints about which modalities should be included in what quality when developing an interactive VE. The senses can be distinguished into *exteroception senses*, which denote the humans' sensitivity to stimuli originating from the outside of the body, and *proprioception senses* that provide feedback about the internal status of the body. The exteroception senses include *special senses*, i.e., senses of vision, hearing, taste, and smell, as well as *somatic senses*, i.e., senses of pressure, heat and pain, together referred to as *senses of touch.* Proprioception senses include the sense of equilibrium, sometimes referred to as the sense of balance, and the *sense of kinesthesia* that denotes the humans' sensitivity to the position and orientation of parts of the human body and limbs relative to other neighboring parts of the body.

A *multimodal interactive system* is a system that relies on the usage of multiple different human communication channels, where each channel is referred to as a *modality* of the interac-

Sensory perception	Sense organ	Modality	Relevance
Sense of vision	Eyes	Visual	>70 %
Sense of hearing	Ears	Auditory	<20 %
Sense of smell	Nose	Olfactory	<5 %
Sense of touch	Skin	Haptic	<4 %
Sense of taste	Tongue	Gustatory	< 1 %

Table 2.1: Exteroception senses and modalities (table according to [Hei92] and [Sil79]).

tion. Moreover, Schomaker et al. distinguish the channels by their direction, i.e., whether information is transferred from the computer system to the human or vice versa ([SNC+95]). The former directed channels are referred to as *output modalities*, whereas the latter channels are denoted as *input modalities*. Table 2.1 gives an overview of the exteroception senses, modalities and their relevance in the human information system.

It shows clearly that the vision system is dominant when humans perceive information from their surroundings. Hence, stimulation of the visual system plays an important role when designing an interactive system or reproducing real-world phenomena in a VE, and thus this issue has become a major focus of research. An interactive system, which only supports one human communication channel, e.g., the visual channel, is referred to as *unimodal* ([SNC+95]).

The second most important sense is the sense of hearing, which is also often considered when developing an interactive system. In this context, the term *bimodal* denotes the usage of exactly two different modalities when interacting with a computer system ([Rai99]). Most interactive systems are at least bimodal since the user views the information displayed on the monitor while interacting with the hand via a mouse or keyboard. Schomaker et al. ([SNC+95]) consider an interaction between humans and computers as *multimodal*, if more than one input- or output-modality, and more than one input device are used. According to Table 2.1, the sense of touch in general does not play a major role for the human information system, but in the real world and especially in HCI manipulation tasks are accomplished primarily by using the hands. In most interactive systems, the senses of smell and taste are neglected, because of their marginal role in the context of HCI and the difficulties in their implementation ([SNC+95]). Proprioception senses, such as the sense of balance becomes more and more important with respect to training system environments, e.g., flight simulators ([BC03]). In a multimodal system, when addressing multiple senses simultaneously, the synchronization of all stimuli with users' actions is essential since insufficient implementation contributes to confusion or to simulator sickness at worst ([Rai99]).

As mentioned above, multimodal interactive systems allow to choose from different input and output channels when transmitting information from the user to the computer system and vice versa. However, the main goal of multimodal interaction is to increase support of the user's abilities by combining different input and output modalities in order to make the use of human-computer interfaces easier and more efficient ([Rai99]). Even though most information is transmitted to the user via the visual channel, the other modalities form an important supplement and extend the interactivity ([BC03]). In the context of HCI Charwat and Schomaker et al. evaluate the visual, auditory and haptic modalities as the most important modalities ([Lat01, SNC$^+$95, Cha92]). Hence, this thesis is focussed on multimodal interaction with special consideration of these three *main modalities*.

2.2.2 Visual Perception

As mentioned in the previous subsection humans perceive most information with their visual sense, and hence the visual representation of a VE is the most important aspect in the context of HCI. Vision essentially depends on light, since the eye receives light reflected by objects. The objects are projected upside down on the *retina* at the back of the eye, where receptors transform the incoming light into neuron signals. As illustrated in Figure 2.1 the *cornea* and *lens* at the front of the eye focus the light to form a sharp image on the retina. The retina contains two types of *photoreceptors*, i.e., highly light sensitive *rods* and less light sensitive *cones*. To allow color vision, there are three kinds of cones, each sensitive to light of different wavelength. These cones are mainly concentrated on the *fovea*, i.e., a small area of the retina on which the image is focussed and from which it is passed to the brain via the optic nerve.

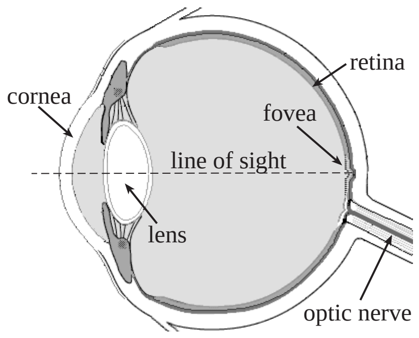


Figure 2.1: Labeled cross section drawing of the human eye (adapted from [Hel95]).

Ideally, the quality of visual feedback generated within a computer system should be equal to the representation in the real world. Unfortunately, todays technology is not capable to achieve this challenge. Hence, when considering the implications on the quality of the visual experience, many compromises have to be accepted.

The human eye has both a vertical and a horizontal *field of view (FoV)* of approximately 180^o by 180^o. Indeed, the vertical range is limited by cheeks and eyebrows to about 150^o, whereas the horizontal FoV is also limited inwards by the nose, and equals 150^o for each eye. When focussed at infinity the total horizontal viewing equals 180^o with a 120^o *binocular* overlap, i.e., the region seen by both eyes of the user ([Hei92]). In comparison, a 21" monitor viewed from the distance of 50cm covers approximately 48^o of FoV, whereas some displays and projection screens use wide field optics that provide up to 140^o of FoV.

Visual acuity is defined as the sharpness of viewing. It is measured as the fraction of a point which spans one *minute of arc (MoA)*, i.e., approximately 0.017^o, horizontally ([CNSD$^+$92]).

Acuity decreases with increasing distance to the line of sight. For instance, for a reasonably lighted object that lies on the line of sight axis, the eye can resolve a separation of one MoA. The area of highest acuity covers only a region of about two degrees around the line of sight, whereas sharpness of viewing decreases rapidly beyond this central area ([Hel95]). Even the best desktop visual displays do not achieve this quality. For instance, a $21''$ monitor with a resolution of 1280×1024 pixels viewed from the distance of $50cm$ supports a resolution of only 2.8 minutes of arc, i.e., 0.047^o.

Temporal resolution of the eye refers to the *flickering phenomena* perceived by humans, when looking at a screen that is updated by repeated impulses as done, for instance, by *cathode ray tube (CRT)*-based monitors. Low refresh rates, especially for large displays and high luminance, cause the perception of flickering. To prevent this effect, a high refresh rate has to be provided, i.e., at least $50Hz$ for small screens and low illumination up to $100Hz$ for large screens and high illumination levels. Todays technology fulfills these requirements, since currently available CRT monitors support at least $85Hz$ refresh rates and higher, whereas modern *liquid crystal displays (LCDs)* are updated constantly and this problem does not occur in this case.

Since this thesis focusses on interaction in three-dimensional VEs, depth perception is essential. To generate depth information the brain extracts cues from the perceived scene. These so-called *depth cues* can be classified into *physiological*, e.g., accommodation, convergence or stereopsis, and *psychological* such as occlusion, object size, motion parallax, linear perspective, or texture gradient ([Lip91]). All of them contribute to depth information as long as no contradictory cues are provided to the user. Usually, three-dimensional virtual environments are projected on two-dimensional displays or projection screens. In the last years, many computer graphic algorithms have been proposed to reproduce these depth cues in synthetically generated images. For this thesis relevant approaches are described in detail in Chapter 3.

2.2.3 Auditory Perception

According to Table 2.1 the sense of hearing has a relevance of about 20% in the human information system, and thus the auditory modality plays an important role in the context of HCI. Sound results from changes or vibrations in air pressure. The human ear receives these vibrations and transmits them through various stages to the auditory nerves.

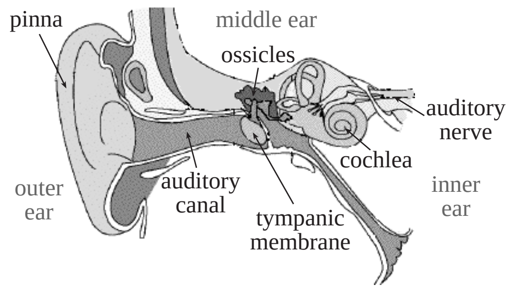


Figure 2.2: Labeled cross section drawing of the human ear (adapted from [Beg94]).

As depicted in Figure 2.2 the ear comprises the three sections, *outer ear*, *middle ear* and *inner ear*. The outer ear consists of the *pinna*, i.e., the outer structure, and the *auditory canal*, along which a sound is passed to the middle ear. The middle ear is a small cavity connected to the outer ear by the *tympanic membrane* and to the inner ear by the *cochlea*. Sound waves are passed through the auditory canal and vibrate the tympanic membrane, that in turn vibrates the *ossicles*, i.e., small bones in the middle ear, which transmit the waves into the inner ear via the cochlea. The waves are passed into the liquid-filled cochlea in the inner ear, which contains delicate hairs that bend due to the vibrations and causes impulses in the auditory nerve ([Gol03]).

Individual sounds can be distinguished by *pitch*, *loudness*, *timbre* and by their *location* in space ([Beg92]). The perceived pitch of a sound is the ear's response to frequency. A low frequency produces a low pitch, whereas high frequencies produce high pitches. The human ear is able to hear frequencies from $16Hz$ to $20kHz$, and it is capable to distinguish changes of less than $1.5Hz$, but for high frequencies the sense of hearing is less accurate ([Dah06]). Loudness is the subjective term describing the strength of the ear's perception of a sound, which is affected by several parameters, including frequency, duration and bandwidth, i.e., measure of the frequency range. Timbre relates to the type of sound and describes the characteristics that allow the ear to distinguish sounds, which have the same pitch and loudness. These capabilities of the human ear can be addressed sufficiently with nowadays sound systems ([KJM03, Beg94]).

The spatial location of a sound is what gives the sound its three-dimensional character. Sound sources are categorized into mono-, stereo- or binaural-, and *3D sound*, also termed as *spatial sound*. The signals of mono sound are equal for both ears. When using stereo or binaural sound two different channels are available to represent a sound source, whereas the distance when recording binaural sound is determined by the distance between the human's ears. To locate spatial sound sources the human's brain processes several different types of data to extract directional and distance information. This data include the shape of the sound spectrum at the eardrum and interaural intensity and time differences, i.e., divergences in sound intensity and time-of-arrival between both ears as well as differences in time-of-arrival between reflections of sound sources. Further cues to support localization of sound sources' positions in space are based on the *proprioception sense*. As mentioned before, this sense of body awareness indicates where the various parts of the body are located in relation to each other. For example, knowledge about the head shadowing or the shoulder reflecting a sound is exploited to localize spatial sounds. Sounds can be located spatially with an error of about 12^o in left/right azimuth angle and 15^o in up/down as well as in front/back elevation angle ([Dah06, ZTW$^+$01]).

Usually, the auditory as well as haptic modalities are used as add-ons to the visual modality in order to support the visual perception. Thus, stereo headphones or multiple audio channels, such as 5.1 or 8.1 dolby surround arrangements, achieve a sufficient quality when generating synthetically spatial sounds ([Dah06, KJM03, Beg94]).

2.2.4 Haptic Perception

The term haptic denotes the sense of touch with both *tactile* and proprioceptive feedback ([Cal05]). Tactile feedback, sometimes referred to as *touch feedback*, is provided by feeling surface geometries, textures, vibrations, and temperature, whereas feeling contours, shapes, inertia and weight of objects enables proprioceptive feedback. Humans are able to sense touch with receptors embedded in the skin, muscles, tendons and joints.

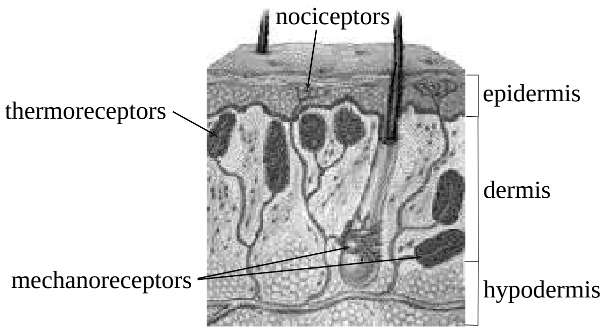


Figure 2.3: Labeled cross section drawing of the human skin (adapted from [Cal05]).

As depicted in Figure 2.3 the human skin is composed of the *epidermis* and the *dermis*. Below these layers lies the *hypodermis*, which is not usually classified as a layer of skin. Furthermore, the skin contains three types of *sensory receptors*. *Thermoreceptors* respond to heat and cold, *nociceptors* respond to intensive pressure and pain. *Mechanoreceptors* respond to mechanical interaction between objects and skin, and convey spatial and temporal information as well as the strength of the force to the appropriate organs. *Rapidly adapting mechanoreceptors* respond to immediate pressure and stop to respond if continuous pressure is applied, in which case *slowly adapting mechanoreceptors* respond ([Gol03]).

Although the entire skin contains such receptors, the spatial resolution varies depending on the *density* and *acuity* of the receptors. The fingertips which provide the highest density of receptors can distinguish two contacts that are at least $2.5mm$ apart, whereas the palm is not capable to distinguish about $11mm$ ([Shi92]). Furthermore, fingertips can distinguish dots of height $1mm$ to $3mm$ on smooth surfaces ([Cal05]). Moreover, the fingertips can distinguish two successive tactile feedbacks of $1ms$ duration at a single location, if they are separated by at least $5.5ms$ ([Jon01]). The resolution of the proprioception sense is defined as the smallest change of the angle of joint that can be detected by a human. It is smallest for the hip (0.2^o) and shoulder (0.8^o) and largest for the finger (2.5^o) and toe (6.1^o) ([Kal93]).

At present, technologies try to exploit the haptic sense to support both tactile feedback as well as *force feedback* ([Cal05]). As mentioned above, tactile feedback refers to information that can be interpreted by the modality of touch. However, tactile feedback does not resist or stop the user's motion, whereas force feedback provides proprioceptive information about surface compliance, weight, and inertia. In the context of haptic computing, *tangible media* refers to the use of real-world objects in a computational setting providing both tactile and force feedback.

2.2.5 Cognitive Psychological Aspects

In the previous sections, fundamentals of the field of perceptional psychology of the human's information system have been described briefly. Humans perceive information by combining their different senses, and cognitive psychology analyzes how humans process and store these information. But also the way humans acquire knowledge, in particular, how humans develop certain skills, or how reasoning works in this context, is a major topic of cognitive psychology. An important concept of cognitive psychology is the *human model processor*, which describes the cognitive process humans go through from perception to action ([CMN83]). Cognitive processing has a significant effect on *interaction performance* defined by ease of learn, ease of use, task completion time and the number of trials required to accomplish a certain task. Thus, when designing an interactive system, only a few cognitive resources should be used in order to allow the user to focus on the actual task. For example, Miller ([Mil56]) describes the strategy of limiting the working memory to increase performance. The working memory can hold only a certain amount of information at a time. If more memory is required to accomplish the desired task, it may result in interference with other parts of the working memory. Hence, interfaces should be designed in such a way that the limited working memory can be used for domain-specific information. An important issue in this context are *affordances*. An affordance is a property of a virtual object, that indicates how to interact with that object. For example, a button indicates to click it with a mouse. Norman ([Nor99]) points out that the usage of affordances supports the user when interacting with virtual objects, since the affordances allow the user to recognize how to interact also with new and unknown objects.

Perceptual and cognitive psychology also considers individual differences in the capability between human beings. McGee ([McG79]) studied the differences of humans' spatial abilities; for example, the classical mental rotations experiment ([CS73]) has shown that humans vary in their abilities to reason spatiality, especially in three dimensions. Thus, when developing interactive systems, designers have to attempt to create interfaces that perform robustly for users with a wide range of capabilities and limitations.

2.3 Computer Systems

In order to understand how humans interact with computers, both actors in the interaction have to be analyzed. In this section, different computer systems are discussed, which provide support to address multiple modalities. Starting with traditional two-dimensional interfaces and their limitations regarding to HCI, more sophisticated approaches that address multiple senses and which allow more advanced interaction concepts are described.

2.3.1 Traditional 2D Interfaces

The first *graphical user interface* (*GUI*) has been proposed by Sutherland in 1963 ([Sut63]). His *Sketchpad* was a unique program developed for the TX-2 computer, which had a nine *inch* CRT that could be interfaced with a light pen. Sketchpad enabled the user to draw and zoom in and out on the computer screen by using this light pen.

Based on this approach during the last decades a set of two-dimensional user interface paradigms have been established using raster devices for 2D output and input devices similar to the light pen. *Input devices* determine the way a user communicates with the computer. Nowadays, the most common human-computer interaction devices are the keyboard, the mouse or other pointing devices in combination with a display. The keyboard and the mouse provide elementary *input events* such as button down or button up, mouse motion or key-press events to control *virtual input devices*, such as the mouse cursor, while a display provides visual feedback as *output event*. This combination has proven to be a very powerful concept for two-dimensional GUIs and has been the de facto standard interface for over 20 years, although even the first prototype appeared already 40 years ago ([EEB67]).

Two-dimensional GUIs are also known as *WIMP* interfaces related to their comprising elements – windows, icons, menus and pointer (sometimes referred to as windows, icons, mice and pull-down menus). WIMPs are the default interface for the majority of interactive computer systems in use today. When using a WIMP-based interface most actions require only fundamental techniques, for instance, a single or double click with a mouse button when the mouse cursor is located at a specific position on the screen. This *point-and-click* interface is not restricted to the usage of buttons, but may also include other WIMP elements, e.g., sliders, text-boxes etc. Usually, two-dimensional manipulations with 2D interfaces are performed by mapping the mouse movements to the mouse cursor on the screen resp. to a selected object located under the mouse cursor's position. However, the point-and-click interface is not tied to mouse-based interactions, and is also extensively used in *touch screen systems*, e.g., teller machines or tourist information terminals.

2.3.2 3D Interaction with 2D Interfaces

The traditional 2D interface is often sufficient, but there is an increasing number of applications in which there is demand for visualization and interaction in three dimensions. In these 3D virtual environments the complexity increases since there is a further axis orthogonal to the screen – the depth axis – which has to be considered when visualizing and interacting. In order to control the interaction with respect to this additional axis, extensions of two-dimensional interaction concepts are required. The two *degrees of freedom (DoF)*, i.e., translation along the x-axis and the y-axis on a projection screen of a display, supported by a standard 2D mouse, may be combined with the keyboard or mouse buttons in appropriate ways. For example, two-dimensional mouse movements could be mapped to planar movements of the virtual input device on the screen, and if a mouse button is pressed in addition it could be mapped to a translation along the depth axis. Alternatively, the scroll-wheel of the mouse (if available) can be used for translations along the depth axis. However, since multiple input paradigms have to be controlled by users, performing 3D interactions with 2D interfaces results in high effort for the user.

Two-dimensional displays also limit the 3D interaction, because in general the user should be able to change the view position from time to time in order to understand three-dimensional structures or to see obscured objects in the virtual scene. For this purpose of changing the view position in the VE, further combinations of mouse buttons, or keys with mouse movements are required causing an extra cognitive effort for the user. HOUDE has shown in [Hou92], that it could be sufficient for certain 3D interaction tasks to use a 2D input device. The choice of input device heavily depends on the characteristics of the application for which it should be used, whereas limiting the available DoF does not necessarily interfere the user. Often it supports the user to reduce the number of simultaneously usable DoF. For instance, if the current task is restricted to slide an object along one axis in space, translation along other axes should not be possible and rotations should be disabled entirely. Such restrictions during an interaction process are called *constraints* and often support a more intuitive way to interact with virtual objects ([SSS01, BH95]).

Although sometimes it is advantageous to use only a subset of the DoF available for a particular input device, generally, the more DoF a device provides the more input options can be used. LIANG and GREEN ([LG94]) describe the problem of insufficient DoF by stating the traditional mouse-based interaction as the bottleneck to 3D geometric design, because it forces the user to decompose a 3D task into a series of one- or two-dimensional subtasks. However, the available DoF have to be used with caution and extraneous DoF should be provided only if their usage proofs a more intuitive facility for the interaction ([LG94]).

2.3.3 3D User Interfaces

To overcome the drawbacks evident in 2D interfaces, many technologies and techniques have been proposed which are more suitable for 3D interaction. When interacting in three dimensions the user has the potential to control more than two DoF. Three-dimensional manipulations of virtual objects can be performed, for example, with six DoF, i.e., three DoF for translations and three DoF for rotations. If a *Cartesian coordinate system* is used, translations can be performed along the x-, y-, and z-axis giving a position, and rotations can be carried out with certain angles around the aforementioned axes. These rotations are called *yaw*, *pitch*, and *roll*, can be used to orientate virtual objects (see Figure 2.4).

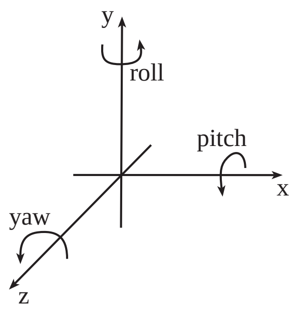


Figure 2.4: Six DoF manipulation in space.

Hence, the position and orientation of a virtual object can be described by a dataset containing six values, which represent the six DoF. To enable an instantaneous and intuitive 3D interaction, this dataset of six values has to be measured rapidly. For this purpose, manufacturers have developed 3D input devices to enable more straightforward interactions. *Global position system* (GPS) transducers report three DoF position with an accuracy of $1m$ to $5m$. But 3D interaction has much more stringent accuracy requirements, for instance, when interacting in a desktop-based arrangement, where accuracy in the range of submillimeters is desired. Special-purpose hardware used to measure real-time changes in position and orientation with appropriate accuracy is called *tracker*. Tracking can be performed with several different technologies, e.g., optical, ultrasonic, magnetic or mechanical approaches. The space-mouse and space-ball are examples for such ultrasonic resp. mechanical desktop-based six DoF input devices ([3DC]).

In general, trackers deliver *absolute data*, i.e., position resp. orientation values, or *relative data*, i.e., change of data compared to the last state. According to [HL95], when evaluating six DoF trackers the most important properties which have to be considered for choosing the appropriate device for a given application are

- *accuracy*, i.e., the measure of error in the reported position and orientation,
- *resolution*, i.e., the smallest change in position and orientation detectable by the tracker,
- *update rate*, which defines the number of measurements per second,
- *latency* that determines the amount of time between the users real action and the beginning of transmission of the report that represents this action, and
- *range*, i.e., the working volume within which the tracker can measure position and orientation.

Beside these properties, also ergonomic aspects have to be considered, e.g., ease of use, size and weight of the device.

Tracking hardware enables to feed spatial information to the system. Besides positioning and orienting virtual objects with six DoF, tracking the user's head enables a *view-dependent exploration* of a VE. Hence, if the head position and orientation of the user is known, the virtual environment can be displayed according to this information. Thus, the user can change the view in the VE by changing his head position or orientation. However, when using monoscopic visual feedback provided by a standard display, both eyes of the user perceive the same image, and important depth information extracted from stereopsis gets lost. In the real world the *interpupillary distance (IPD)* (about $6.5cm$) is fundamental for stereopsis to enable human beings to interpret distances to objects. The user perceives points of the real world shifted horizontally with the right and the left eye, because of their different positions in both eyes. This shift is called *image parallax* and has to be emulated in order to enable humans to interpret such binocular depth cues in the virtual world. Stereoscopic display technologies allow the user to experience the scene in stereo by providing a pair of images. Showing each eye a slightly different view of the scene according to the eye distance makes the human's perception system to extract the disparity in the two views to create a stereoscopic image. When two displays are used, each presents one image to the corresponding eye. When a single display is used, the two images can be *time-sequenced*, i.e., the image for the left eye is displayed first and the image for the right eye is displayed afterwards, *filtered*, i.e., both images are displayed superimposed, or *spatially sequenced*, i.e., the images are displayed entirely next to each or they can be displayed interlaced. Depending on the stereoscopic technology the separation of both images can be performed, e.g., by using active shutter stereo glasses, passive stereo glasses, or by using autostereoscopic displays.

The usage of 3D input devices in combination with stereoscopic visualization enables an improved 3D interaction in comparison to standard desktop-based environments. Besides 3D interaction with objects, many more input paradigms are involved in real-world interactions since manipulations humans perform in the real world are not restricted to six DoF. For example, humans can use multiple finger combinations with different forces to grab and deform objects. Furthermore, interactions can be initiated by means of articulation or gestures ([Lat01]). Hence, it is desired to immerse the user more into the virtual environment by addressing these multiple input options.

2.3.4 Virtual Reality

The term *virtual reality (VR)*, sometimes referred to as *artificial reality* or *simulated reality*, has been used to refer to the notion of an advanced computer generated experience. The virtual environment responds to the input of the user and provides *real-time* interactivity. Real-time interactivity requires that the computer is able to detect the user's input and modify the VE instantaneously, which corresponds to refresh rates of at least 15 *frames per second (fps)* with which the scene is rendered ([AMH02]). Since in most VR systems auditory and haptic modalities are used in conjunction with the visual modality it is important that all stimuli are synchronized with the users' actions and with each other.

Interactivity and the usage of impressive 3D computer graphics contribute to the feeling of *immersion*, i.e., the feeling of being part of the VE. To further increase the immersion, VR technologies support several human sensorial channels, i.e., interactions are not restricted to the visual sense, but the user can also hear, touch and feel the VE.

Burdea and Coiffet ([BC03]) define VR as follows:

> *"Virtual reality is a high end user-computer interface that involves real-time simulation and interactions through multiple sensorial channels. These sensorial modalities are visual, auditory, tactile, smell, and taste."*

However, VR is not limited to advanced user interfaces. Moreover, VR applications help to solve real-world problems since VR technologies and techniques provide more immersive interactions with VEs and thus provide a better insight into complex datasets leading to an improved comprehension of complicated structures.

A taxonomy proposed by Zeltzer classifies VR into the broad field of graphic simulation systems ([Zel92]). This classification includes both VEs as well as more conventional computer animation and graphic simulation systems. The taxonomy model is based on a coordinate system involving three axes representing *interaction*, *presence* and *autonomy* (see Figure 2.5). In this context interaction refers to the ability of the user to affect and modify the virtual scene. Presence is a synonym for immersion, i.e., the degree to which input and output channels of the system and the user are matched, whereas autonomy measures the quality of the system to act and react to generated events and stimuli. In this model applications that are mapped to the area around the upper right front corner $(1, 1, 1)$ are referred to as VR applications, whereas, for example, interactive simulation environments are mapped in the region at the back plane $(0, 0, 0)$ to $(1, 1, 0)$ of the cube. Hence, VR applications provide digital worlds that involve a high degree of interaction and autonomous behavior of the VE in order to support the immersion into the virtual world by addressing multiple senses.

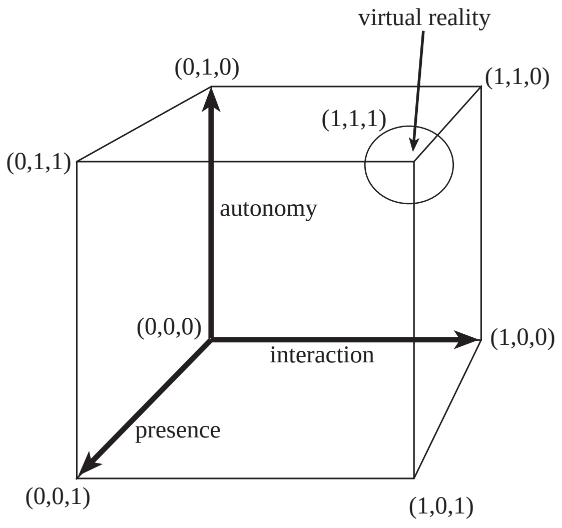


Figure 2.5: Taxonomy of interactive graphics systems (adapted from [Zel92]).

Before giving an overview about current VR technologies, the next subsection summarizes how VR devices and techniques have evolved during the last 45 years.

Brief History of VR

In 1962, Heilig issued the US Patent no. 3 050 870 for the *Sensorama* invention, which was the first video arcade game, providing three-dimensional color video, stereo sound, motion, aromas, wind effects and a vibrating seat. Three years later Sutherland proposed the ultimate approach to a VR system as an artificial world construction concept that involves interactive graphics, force-feedback, sound, smell and taste ([Sut65]). This work is a theoretical description of an ultimate virtual reality scenario, in which the user is completely immersed with all senses into the virtual world.

In 1968 Sutherland has constructed a device considered as the first *head mounted display (HMD)* with appropriate head tracking ([Sut68]). This HMD technology used two rather large displays, which provide a stereo view that was updated according to the user's head position and orientation. Further development has been primarily driven by the US military, which has used VR technologies, in particular HMDs, to support education of pilots in conjunction with flight simulators. In the middle of the 1980s the VPL Research company manufactured the first commercially available VR hardware devices, e.g., data gloves (1985) and HMDs (1988).

At the beginning of the 1990s the development in the field of VR accelerated and the term virtual reality itself became extremely popular. In 1992 the *CAVE* (CAVE Automatic Virtual Environment) has been presented as virtual reality and scientific visualization system. Instead of using HMDs it projects stereoscopic images on the walls of a cubic room. Such projection-based displays provide each eye an independent image either time-sequenced or time-parallel using shutter resp. filter technologies. These projection-based display systems assure superior quality and resolution of stereoscopic images, and a wider field of view in comparison to most HMD-based systems. Hence, as denoted in [BKLP04] nowadays most VR systems are based on projection-based hardware systems, such as the CAVE, single projection-walls or workbenches (see Chapter 6), i.e., tabletop-based projection screens.

Levels of Immersion

VR systems generate sophisticated sensory impressions that are delivered to the human senses. The type and the quality of these impressions determine the *level of immersion*. Ideally, information should be presented to all of the user's senses with high-resolution, high-quality and consistent over all the representations ([SSC01]). Moreover, the environment itself should react realistically to the user's actions.

In the literature VR systems are often grouped according to the level of immersion they provide to the user ([BC03, MG96, IS94]):

- **Desktop VR systems**, sometimes referred to as *Window on World* (WoW) systems, are the simplest type of a VR system. Generally, only conventional monitors are used to display the VE either monoscopic or stereoscopically. These systems are often combined with auditory feedback. Usually, in desktop VR systems only standard desktop devices, e.g., mouse and keyboard, are used as input devices.

- **Fish tank VR systems** enhance desktop VR systems. These systems support head tracking and view-dependent exploration. Usually, in fish tank VR systems a conventional monitor in combination with LCD shutter glasses for stereoscopic viewing is used. As in desktop VR systems these fish tank VR systems are often combined with auditory feedback. These systems often enhance the input paradigms of standard desktop-based environments by six DoF devices, such as a space-mouse or a trackball.

- **Semi-immersive VR systems** provide more immersive VR in terms of display technology and multisensory output. Usually, semi-immersive systems exploit large projection displays that support stereoscopic viewing of the scene according to the user's head position. The interaction is performed by the usage of VR input devices, e.g., data gloves, which enable natural manipulation of virtual data. Usually, these systems are enhanced by audio and haptic sensory interfaces.

- **Immersive VR systems** enable complete immersion in virtual environments by the usage of HMDs or CAVEs, head phones and data gloves. Hence, the audiovisual perception exclusively takes place in the virtual world.

While desktop VR and fish tank VR systems are more related to 3D user interfaces, semi-immersive and immersive VR systems are typical VR setups widely used in research and industrial VR laboratories. This thesis concentrates on semi-immersive VR systems, although the presented approaches are not restricted to them.

2.4 Interaction Concepts

In the previous two sections human factors and different computer systems have been described. In this section, the way how users exploit computer systems to perform certain tasks is discussed. Before the communication between both is analyzed, basic terms of interaction are introduced and fundamental models, which describe the dialogue between the user and the system, are presented. Finally, generic interaction tasks addressed within the scope of this thesis are sketched.

2.4.1 Basic Terms of Interaction

Traditionally, the purpose of an interactive system is to aid users in accomplishing *goals* or *objectives* in specific application *domains*. Domains define areas of expertise in real-world activity, e.g., medical diagnosis or urban planning. *Tasks* are operations used to manipulate the concepts of a domain to achieve certain objectives. An *intention* is a specific action required to meet the objective ([DFAB98]). *Task analysis* involves the identification of demands in terms of objectives, tasks, and intentions for using an interactive system in a certain domain. The interaction between the user and the system takes place by means of communication, for which different languages are used. The *system language* describes computational attributes of the domain relevant to the system's state, whereas the *user language* describes attributes relevant to the user's state ([DFAB98]).

Norman introduced a model of interaction – the *execution-evaluation cycle* – which distinguished two major phases: *execution* and *evaluation*; each can be divided into substages again resulting in seven different stages ([Nor98]). First, the user establishes an objective (1) and forms the intention (2). In the next step, the user has to specify (3) and execute (4) the action sequence. Afterwards, the user perceives the system state (5) and has to interpret it (6). Following, the user evaluates (7) the system state with respect to the objectives and intentions in order to decide about further actions.

As denoted above, user and system use different languages to communicate which may potentially lead to confusion. Norman called the discrepancy between the user's formulation of the action sequence to achieve an objective and the actions allowed by the system the *gulf of execution.* Moreover, he denoted the difference between the physical representation of the system state and the expectation of the user as *gulf of evaluation.* Only if these differences are marginal, an interaction can be performed effectively.

2.4.2 General Interaction Cycle

Although the execution-evaluation cycle model helps to understand HCI, it considers only the user's view of the system. Abowd and Beatle address this problem and provide a more realistic description of HCI by means of incorporating also the system's communication with the interface ([AB91]). This so-called *general interaction cycle* is illustrated in Figure 2.6.

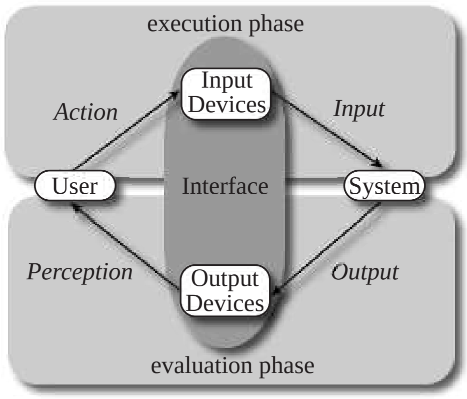


Figure 2.6: General interaction cycle with transitions (adapted from ([AB91]).

It consists of four major components: the user, the system, the input devices and the output devices. The user and the system communicate via the interface which is formed by the input and output devices. In the *execution phase* the interaction cycle is initiated by the user formulating an objective and a task to achieve that objective. In the next step, the user executes an action in order to accomplish the task at the interface via the input devices. These actions are transferred as input to the system. Hence, the system state changes with respect to these actions. When the execution phase of this cycle is finished, the evaluation phase begins. After processing the actions the system posts output to the interface via the output devices. Finally, the user observes and interprets the output, and he reviews the results relative to the initial objective. Depending on whether the goal has been achieved new interactions can be initiated, or the cycle has to be traversed again.

2.4.3 Layered Interaction Concepts

To improve the dialogue between the user and the system many approaches have been proposed, which support different levels of interaction within an application ([ABF$^+$94, BH93, Nie86]). Figure 2.7 shows a general reference model for the actions performed by the user in the execution phase illustrated in Figure 2.6.

The bottom level is defined by the *hardware layer*, sometimes referred to as *physical layer*. This layer incorporates *input devices* and corresponding *input events*. When performing two-dimensional interactions these devices and events are given, for example, by mouse or keyboard devices resp. button-press or key-press events. In the case of three-dimensional interactions these

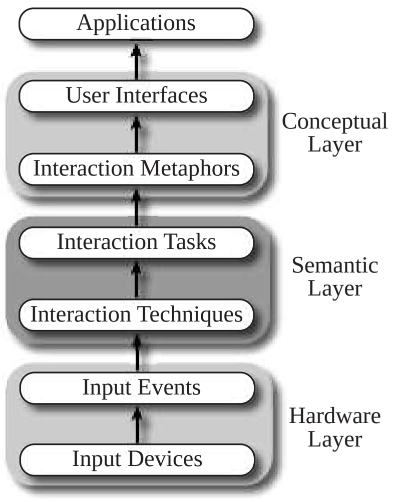


Figure 2.7: Layered model of interaction (adapted from [ABF$^+$94, BH93, Nie86]).

devices may be a space-mouse or data glove devices, causing six DoF motion or gesture events.

The next level is given by the *semantic layer*, which incorporates the functionality of an interactive system, i.e., interaction tasks which can be performed within the system by the usage of certain *interaction techniques*. Typical interaction tasks are translation, rotation, or navigation, which can be performed by using several interaction techniques, e.g., clicking or dragging in 2D, and pointing or grabbing in three-dimensional environments. The communication embodied in such interaction techniques can be divided into *syntactic*, *lexical* and *alphabetic levels*, which define the grammatical structure of a sequence of tokens, the structure of the tokens, and the primitive symbols used to articulate the semantic concepts ([Shn97a]). For example, consider the selection of a desktop icon which is a semantic level task. The syntax to perform such a task is, for example, given by the two lexical actions: move the mouse cursor to the position of the icon and press the left mouse button. The corresponding alphabetic level actions are to move the mouse cursor to pixel (x, y) overlapping the icon's position on the desktop, and to press the left mouse button at that location.

Interaction tasks can be realized with a variety of combinations of different interaction techniques. Many of these *basic interaction techniques* are based on an object-action or action-object paradigm in which a user identifies an object and an action to be applied to that object or vice versa ([Shn97a]). Basic interaction tasks are indivisible, i.e., decomposing them into smaller units would destroy their meaning in the context of the application. More complex interaction tasks can be performed by composing basic interaction techniques to *composed interaction techniques* ([JvDFH92, FD82]).

The next layer of the model is the *conceptual level*, which incorporates the main concepts of an interactive system as seen from a user's point of view. Foley and Van Dam refer to this model as *user model* or the *mental model* ([FD82]). Accomplishment of many interaction tasks puts a high cognitive effort on the user, since the objects and their manipulations are virtual and often not related to real-world interactions. For this purpose, *interaction metaphors* support understanding of abstract aspects in a complex correlation by representing these complicated techniques and tasks by well-known procedures. Interaction metaphors as part of the *user interface* have been well-established in two-dimensional GUIs, e.g., through the *desktop metaphor* ([Com87]). Since interaction metaphors for VR environments are in the main focus of this thesis the next subsection discusses interaction metaphors in more detail.

2.4.4 Interaction Metaphors

According to Carroll, Mack, and Kellogg ([CMK88]), an interaction metaphor, sometimes referred to as *interface metaphor*, is defined by interface concepts that exploit specific knowledge that users have from other domains. Thus, developers can incorporate interaction metaphors to control the complexity of an interface by using preexisting actions, procedures, and concepts. If a metaphor is chosen properly, objects within the interface imply natural and intuitive interaction. In that case, even novice users know how to interact with specific objects, and they are able to predict the effects of their actions.

Barr, Biddle and Noble provide a taxonomy of interface metaphors ([BBN02]). This taxonomy involves the following approaches.

- *Orientational metaphors* explain concepts in terms of space. They are not only used for navigation, but also for quantification. For example, when using vertical sliders, moving up generally means more and down means less.
- *Ontological metaphors* use experience with physical objects to provide a better understanding. For instance, treating data files as objects enables to assign them size and location.
- *Structural metaphors* characterize the structure of one concept by comparing it to the structure of another one. The desktop metaphor with the analogy between files and folders and their real-world equivalents belongs to this class of metaphors.

When a metaphor explains one concept in terms of another, *metonymy* is the usage of one entity to refer to another that is related to it. For instance, metonymy is used in GUIs if icons are used to represent files.

The usage of metaphors may also have drawbacks. For instance, extensive usage of metaphors forces the user to understand the system only in terms of the metaphors. Hence, when designing new metaphors, it must be ensured that all metaphoric information the metaphor implies about the application are clearly indicated. Otherwise metaphorical mismatches can occur when using an interaction metaphor. For example, the metaphor may contain concepts that are not supported by the computer system, or the computer system may support actions that are not presented in the metaphor. Furthermore, the computer system may behave in a way which conflicts with the behavior predicted by the user. Since canonical mapping is subjective to users or groups of users, the selection of an appropriate metaphor is a challenging task in the design of a user interface.

2.4.5 Taxonomy of Interaction Tasks

After fundamental concepts of the interaction process have been described in the previous sections, this subsection focusses on interaction tasks, which are *generic* in the sense that they are not related to specific domains and that they are widely used in many interactive systems.

Early efforts in human-computer interaction sought to identify generic tasks that appear repeatedly in human-computer dialogs. For example, most interactive systems but also sophisticated VR systems require interaction tasks to solve problems including the navigation within a virtual scene, e.g., to zoom in or out, identification of objects, movement of objects from one place to another or modification of the appearance of an object. FOLEY, WALLACE, and CHAN ([FWC84]) discovered that user interface transactions are composed of the following generic interaction tasks:

1. **Selection**: Choosing virtual objects or items from a set of alternatives.
2. **Positioning**: Specifying a position within the virtual space, e.g., to translate virtual objects.
3. **Orienting**: Specifying an angle or three-dimensional orientation, e.g., to rotate virtual objects.
4. **Path**: Specifying a series of positions and/or orientations over time.
5. **Quantify**: Specifying an exact numeric value.
6. **Text**: Entry of symbolic data.

One of the goals of VR research is to provide a taxonomy of interaction tasks and techniques with special consideration to VR environments Researchers have created detailed and slightly different taxonomies ([Bow99, BH97, SNC$^+$95]). BOWMAN ([Bow99]) proposes one of the most common taxonomies that divides interaction tasks into four main categories: Navigation, Selection, Manipulation, and System Control, which might be considered as a combination of selection and manipulation. Navigation subdivides into the tasks of *wayfinding* and *traveling*. Wayfinding denotes aspects of figuring out where to go and planning how to get there, traveling is the technique of actually getting there. In this thesis, instead of navigation the more abstract interaction task of *exploration* is considered. Exploration involves viewing the scene from different view positions or with different view characteristics. Selection is the specification of a target for the interaction, e.g., one or more virtual objects, or menu entries. Manipulation of virtual objects refers to the modification of a particular object's attributes, e.g., color, position, orientation or scale. SHNEIDERMAN introduces the term *direct manipulation*, which denotes manipulations by the usage of continuous representations of objects and actions and rapid feedback, instead of using menus or prompting numeric values before applying an action ([Shn97b, Shn97a]). System control focusses on selection and manipulation of system properties to change the state of a VR application.

Since the metaphors proposed in this thesis are *universal*, i.e., they are not restricted to specific objects or domains, they can be extended to "Quantify" and "Text" tasks, for instance by selecting objects that represent alpha-numeric values.

The technologies and the potential VR provides for HCI open up new vistas for the interaction between human and computer, but also pose challenges related to the handling of unfamiliar interaction devices and concepts ([BH99]). Only few VR interaction metaphors such as direct manipulation via the virtual hand metaphor (see Chapter 4) are known ([Ter05, BKLP04, BH97]), further metaphors and paradigms for VR interaction are required. This thesis provides new ideas to this research area and introduces new interaction concepts for VR-based interactions.

Chapter 3

A Generic Virtual Reality Software System

In Chapter 2, fundamental concepts for the interaction between a user and a system have been discussed. In this section a framework is presented that allows the transfer and rapid deployment of interaction concepts into VR-based applications. As pointed out in the previous chapter, VR systems may use several input and output channels in order to make interaction in virtual environments more intuitive and to increase the user's immersion into the virtual world. When developing VR-based applications, developers should be able to focus on modeling advanced interaction and system behavior instead of having to concentrate on hardware connections and rendering issues. Many systems and tools for developing VR-based applications have been proposed to achieve this goal. However, no de facto standard is available. In this chapter, an approach for a generic virtual reality software system to handle such VR-related issues is discussed.

The next sections briefly explain a fundamental architecture used for interactive applications and describe building blocks for a generic VR software system. Section 3.2 discusses the demands on and related work about VR software systems. Section 3.3 describes the computer graphics system VRS, the *Virtual Rendering System*, which has been extended by the VR software system component *Virtual Reality VRS* (VR^2S) within the scope of this thesis. The system architecture and the integration of some essential VR-related components is described in detail in Section 3.4.

3.1 VR Software Systems

This section sketches the building blocks for a generic VR software system. An architecture that is widely used for modeling interactive systems is discussed first.

3.1.1 Model-View-Controller

When implementing an interactive system, in particular a VR software system, it is important to subdivide the system such that modifications to one component can be made with minimal impact on the others. The *model-view-controller (MVC)* is an architecture that separates the data model of an application, the user interface, and the control logic into three distinct components ([GHJV93]). Constructing an application using the MVC architecture involves three classes of components.

- **Model**: This is the domain-specific representation of the information on which the application operates.
- **View**: The view component displays the model in a form suitable for the interaction, e.g., a graphical user interface.
- **Controller**: The controller component responds to events, such as user actions, and invokes changes on the model and possibly the view component.

When using a MVC architecture control flow generally works as follows. The user interacts with the user interface in a particular way, e.g., the user presses a button as part of a GUI. The controller component handles the input event obtained from the user interface, usually via a registered *handler* or *callback*. Instead of waiting for certain commands, the system is in an *event loop*, i.e., it looks repeatedly for information to process, e.g., keyboard or mouse events. When an event occurs a default or user-defined *trigger function* processes the event. *Event-driven* systems typically contain a number of event handlers that are called in response to external events, and a *dispatcher*, which calls the event handlers. Such a handler or callback accesses the model component and updates it according to the user's actions. The view component uses the model component to generate an appropriate user interface. The MVC paradigm introduces the controller object in between the view and the model component to communicate between the both. Further information about MVC concepts can be found in [GHJV93].

The MVC architecture is widely used as design pattern for interactive applications, and important concepts presented in this thesis are based on this paradigm.

3.1.2 Generic Architecture of VR Systems

As discussed in Chapter 2, virtual reality refers to the notion of an advanced computer generated experience. To realize such an experience the communication between several VR-related components has to be organized. VR systems enable and coordinate the interaction between these highly interactive components. Figure 3.1 shows the main classic components of a VR system ([BC03]).

This section concentrates on the realization of a generic *VR engine* and its relation to input and output devices. While the VR engine represents the key component of a VR software system, I/O devices define the VR hardware system, e.g., tracking systems, data gloves, HMDs or CAVEs. The VR engine handles communication with the input and output devices and

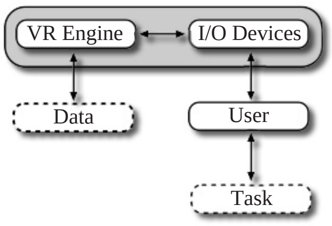


Figure 3.1: A generic VR system (adapted from [BC03] and [Vin95]).

performs the required real-time computations to update the state of the virtual world, i.e., to receive the information from the I/O devices and to render the results to the used multimodal output systems.

3.1.3 The Rendering Pipeline

The term *rendering* is generally associated with the process of converting 3D geometric models populating a virtual world into a 2D image displayed to the user. In the context of VR systems, the meaning of rendering is extended to include other sensorial modalities, such as auditory and haptic. Hence, rendering in the context of VR systems describes the transformation process from an arbitrary virtual entity defined by the application developer to the representation on corresponding output displays. During rendering each object is handled separately to enhance performance by enabling processing of further objects before the rendering of previous ones is finished. This is realized in current rendering systems by using a *rendering pipeline.* Numerous objects going through the pipeline sequentially are processed simultaneously by the different *stages* of the pipeline. Because of the sequential pipeline processing of virtual objects, computation of global effects depending on interactions between virtual objects, e.g., shadows or (visual or acoustic) reflections requires additional passes of the corresponding objects through the pipeline.

In the context of this thesis rendering pipelines are considered for the main three modalities, i.e., visual, auditory and haptic modality. In contrast to graphics pipeline concepts, haptics and acoustic rendering pipelines have no uniform standardized architecture, which reflects the rapidly evolving state of today's technology.

Graphics Rendering Pipeline

Since the OpenGL *application programming interface (API)* ([Shr04]) has evolved as de facto standard, the pipeline concept for graphics systems is explained on base of OpenGL. The *graphics rendering pipeline* consists of three conceptual stages: the *application stage*, the *geometry stage* and the *rasterizer stage* ([AMH02]) (see Figure 3.2).

In the application stage those graphics objects are determined which are further processed by the following stages of the rendering pipeline. Since this determination is not implemented in graphics hardware but in software, the application developer has full control over this stage

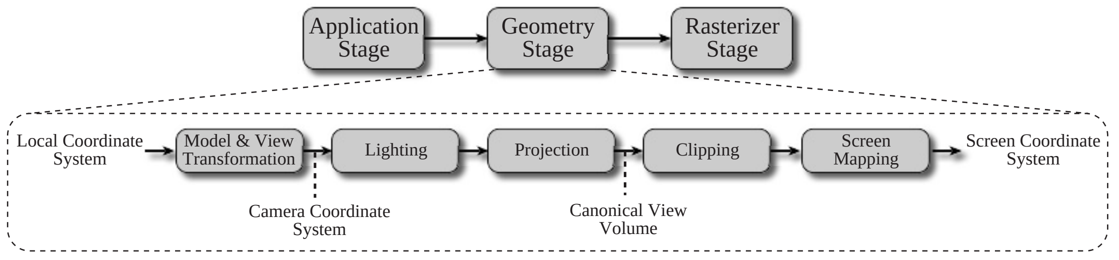


Figure 3.2: Conceptual stages of a graphics rendering pipeline and functional substages of the geometry stage ([AMH02]).

of the rendering pipeline and can define algorithms and techniques to influence the number and type of objects further processed. For example by using *culling techniques*, the number of objects passed through the pipeline can be reduced. View frustum culling discards objects outside the region projected on the screen, occlusion culling removes objects invisible to the user since they are occluded by other objects. After the user has defined in the application stage which objects continue down the pipeline, geometric transformations are applied to these remaining objects in the geometry stage.

Graphics objects are defined by *vertices* in a *local coordinate system*, sometimes referred to as *model coordinate system*. These vertices are transformed via the *world coordinate system* into the *eye coordinate system*, sometimes referred to as *view* or *camera coordinate system*, by applying a composition of the *model* and the *view transformation*, called *modelview transformation*. In the eye coordinate system an object is defined in relation to the camera which is at the origin of the coordinate system. When the objects are given in eye coordinates, lighting is computed to determine for each graphics object the color of its vertices. Afterwards, the scene content is transformed such that the visible graphics objects lie in the *canonical view volume*. Two different types of projections are used in current real-time rendering systems: *perspective projections* and *orthogonal projections*. A perspective projection matches the perspective of the human visual system by using a center of projection and a projection plane. Thus, objects farther away from the camera appear smaller and parallel lines may not remain parallel when projected perspectively. An orthogonal projection is determined by a direction of projection and a projection plane. Parallelism is retained, and the sizes of objects do not vary with changing distance to the virtual camera. The canonical view volume is given by a cuboid volume centered at the origin. After transforming the objects in the canonical view volume, *clipping* against this volume can be performed very efficiently. When clipping is finished the remaining objects are projected to the screen area, where the final image is to be displayed. This *screen mapping* into the *screen space coordinate system* is done by applying the *viewport transformation*. The viewport transformation scales the scene that is given in the canonical view volume to fit the target screen area ([AMH02]).

Finally, when all graphics objects are given in screen space, they can be *rasterized* by assigning correct colors to the corresponding pixels on the screen. In this rasterizer stage, this pixel information is rendered into the *frame buffer*, which is a rectangular array containing pixel data and has the same size as the target screen area. The frame buffer usually consists of several layers, e.g., a *color buffer*, which stores colors in the RGB-mode, and a *depth buffer*, which contains the pixels' depth values. Rasterizing objects directly into the visible frame buffer has the following drawback. When displaying a series of images, e.g., an animation or an interactive scene, permanent clearing of the frame buffer and rasterization of the scene content is required. This may create flickering phenomena and disturb the perception of smooth movements on the screen. Thus, commodity hardware provides a second frame buffer, called *back buffer* and supports rendering of the current frame in the background. After the rendering of the image into the back buffer is finished, the back buffer's content is swapped with the visible *front buffer*, and then the next frame can be rendered into the back buffer. For time-sequential stereo rendering, todays high-end graphics cards provide two additional buffers to enable eye dependent rendering, i.e., a *front left*, *front right*, *back left* and *back right buffer*.

Acoustics Rendering Pipeline

Acoustic rendering, sometimes referred to as *auditory or sound rendering*, describes the process of forming the composite soundtrack from different *sound objects*. In the context of VEs, sound objects are treated as time dependent acoustic signals defined in the virtual world, which can be perceived by the user via an auditory display system. In terms of time, sound objects can be either bounded or semi-infinite, e.g., a repeated waveform. Furthermore, sound objects can be defined as point sources somewhere in the VE, where they stay fix, or they can be associated to virtual objects and change their position, when the associated object's position is modified. With these acoustic signals the auditory display produces the illusion of a sound source emitted from a certain position in a particular acoustic environment perceived by a *virtual listener*, i.e., the analog to the virtual camera in the graphics pipeline.

By controlling the listener, the application developer controls the way the user experiences the virtual world auditorily, since the virtual listener is defined by the sampling point and orientation, and other parameters, e.g., velocity, that affect the output stream ([Hie05]).

To enhance the process of sound rendering a pipeline concept similar to the graphics pipeline concept has been introduced in [TH92].

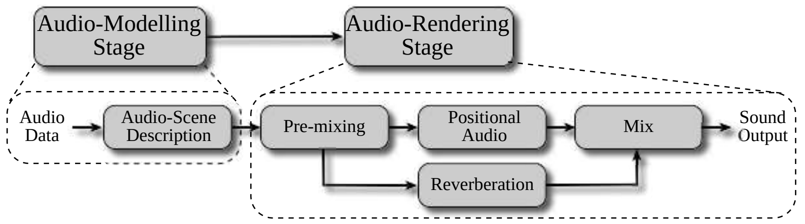


Figure 3.3: Conceptual and functional stages of the acoustic rendering pipeline (adapted from [Hug05] and [TH92]).

This pipeline is illustrated in Figure 3.3. This *acoustic rendering pipeline* consists of two conceptual stages: the *audio-modeling stage* and the *audio-rendering stage*. In the audio-modeling stage the application developer initializes audio sources and describes the audio scene by assigning characteristics such as volume or direction to user-defined sound sources, which can be, for instance, recorded or synthesized. These sound sources can be attached to virtual objects or they can be assigned a fixed position somewhere in the VE.

If a sound object is to be displayed, i.e., played with a speaker, first the sound object is further processed by the audio-rendering stage. In the first *pre-mixing substage* corresponding sound operations are applied. These pre-mixing operations include variable delay lines, resampling and filtering of sound objects, e.g., to support occlusions, directivity functions and distance attenuation. The resulting sound objects can be transferred either to the *positional audio stage* or to the *reverberation stage*. In the positional audio stage the acoustic signals are computed as a function of the distance and orientation between the virtual listener and the sound source as well as the radiation characteristics of the source and the properties of the acoustic environment and then transferred to the auditory display. In the *mix stage*, sound sources from the positional audio and reverberation stages are mixed to generate reverberation effects, if specified by the application developer.

Finally, the resulting sound objects are passed to the sound output devices, e.g., headphones or dolby surround speakers.

Haptics Rendering Pipeline

Haptic rendering denotes the representation of haptic feedback submitted by a haptic interface ([Cal05]). The computation of haptic feedback considers object-object collisions and collisions between virtual input devices and virtual objects. Moreover, these algorithms ensure that a certain haptic device correctly renders resulting forces. The pipeline concept is also applicable for haptic rendering and can be implemented through a multistage haptic rendering pipeline ([SCB04, Pop01]) illustrated in Figure 3.4.

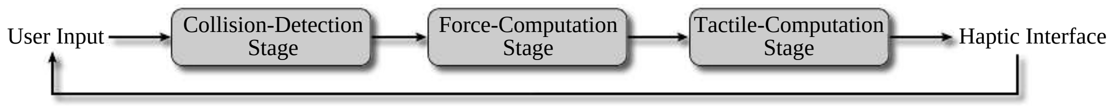


Figure 3.4: Conceptual stages of the haptics rendering pipeline (adapted from [Pop01]).

Three main stages compose a typical haptic rendering pipeline. In the *collision-detection stage* collisions between virtual objects and virtual input devices in the virtual environment are detected. This yields information about the position and the time of a collision, and ideally includes other information about the collision, such as penetration depth, indentations, contact area etc. Usually, the application developer specifies collision-detection algorithms and the physical characteristics of the virtual objects. Such characteristics of virtual objects include surface smoothness, weight, temperature etc. The potentially colliding structures are passed to the next stage.

The *force-computation stage* defines the interaction force between virtual input devices and virtual objects in case a collision is detected. This force approximates as closely as possible the contact forces that would usually occur during contact between real-world objects, and therefore the computed collision forces are based on various physical simulation modes ([Pop01]). These so-called *force-response algorithms* typically use the virtual input device's position, the positions of all objects in the virtual environment, and the collision state between input device and virtual objects. Results of this evaluation process are force and torque vectors that are fed to the interfaces of the corresponding output devices. This stage also involves *force smoothing* and *force mapping*. Force smoothing adjusts the direction of the force vector in order to avoid sharp transitions between polygonal surfaces, whereas force mapping projects the calculated force to the characteristics of the particular haptic display system, e.g., the *PHANToM* device ([Cal05]).

Finally, in the *tactile-computation stage*, the resulting haptics are rendered for the simulation. The computed effects, such as vibrations, temperature etc., are added to the force vector and are sent to the haptic display system.

3.2 Concepts of VR Software Systems

In VR systems the main focus is on interaction. Interaction combines adequate representations of the virtual environment with the manipulations available in the VE. Due to the availability of special hardware, in particular intuitive VR input devices, applications in this context are highly-interactive and enable miscellaneous interaction concepts.

Within dynamic and responsive computer generated VEs, multimodal interactions are typically realized by using special VR hardware. As denoted in Chapter 2, these devices such as data gloves, wands and speech recognition systems enable users to navigate in and to interact with the virtual world more immersively since multiple senses are addressed during the interaction process. In addition, immersive display technologies, e.g., stereoscopic projection systems and HMDs, in combination with various multisensory output paradigms for auditory and haptic feedback improve immersion into the virtual world. Another important aspect of VR systems is real-time performance. Response time and update rate of the system need to be sufficient to avoid latencies.

When designing VR applications developers should focus less on rendering issues and implementing interfaces to devices but rather on interaction and simulation in order to make VR worlds become more interactive and responsive. For this purpose, *VR software systems*, sometimes referred to as *VR software toolkits*, facilitate an abstract view of the system and the input as well as output devices by providing higher-level development libraries to reduce the effort needed for creating and rendering VEs. Since low-level graphics APIs such as OpenGL and DirectX address basic rendering issues, VR research can focus on more advanced topics, e.g., to further extend multisensory interfaces and to advance basic interaction techniques by providing intuitive metaphors.

3.2.1 Requirements on VR Software Systems

In [BJ98] Bierbaum and Just discuss requirements on VR software systems and point out three major demands in terms of *performance*, *flexibility* and *ease of use*.

1. **Performance**:
   Effective immersive VR environments require high frame rates and low latency. Hence, VR software systems should exploit all available resources of a system and should make use of special graphics hardware to achieve interactive frame rates.

2. **Flexibility**:
   The VR system should be adaptable to arbitrary hardware and software configurations. Developers should not need to reimplement an application in order that it works well with new configurations.

3. **Ease of Use**:
   A VR software system should be easy to configure and to learn. Simple applications must be easy to implement; moreover, the system's underlying complexity should be hidden as much as possible from the application developer.

However, often trade-offs must be made in meeting these demands in a VR software system. For example, a flexible design of a VR software system may result in a loss of performance, since hardware specific options may not be supported. Further demands on VR software systems include especially hardware-oriented issues such as support for multisensory output, various input paradigms as well as device independence. Also multi-user support for collaborative interaction should be possible within a VR software system. Furthermore, VR software systems involve additional technical requirements such as extensibility, modularity etc.

3.2.2 Overview of current VR Software Systems

Many VR software systems and VR toolkits have been proposed to support the development of VR applications. For example, users of *VPLs Body Electric* ([AKO95]), which is a real-time simulation system for VR, can specify relations between virtual objects and input or output devices in a dataflow diagram editor. This dataflow approach can also be found in a variety of similar systems such as SGIs Open Inventor ([Gro05]). However, a dataflow approach does not support program modularity ([BJH$^+$01]).

Alice ([PBC$^+$95]) is a rapid prototyping system for creating interactive computer graphics applications without requiring a strong technical background. Since Alice has not been designed to allow complete control over geometry at the polygon and vertex level for large amounts of data, the library is unsuitable for scientific and complex data visualization.

CAVElib ([CN95]), which is based on *OpenGL Performer* ([SGI04]), provides a low-level API for creating VR applications, in particular for projection-based systems. Bierbaum and Just ([BJ98]) mention that the CAVE library was not intended as a long-term solution for VR development, and thus its API is often difficult to extend in backwards-compatible ways. In addition, the library is inadequate for non projection-based VR systems, such as HMD-based systems, which require different projection techniques.

AVANGO ([Tra99]) extends the concepts of OpenGL Performer in order to support the development of multisensory VR-based applications. Since AVANGO is targeted at high-end SGI platforms, it is not suitable for other platforms. The modular system *Lightning* ([BLRS98]) is an object-oriented system for creating and developing VR-based applications. Similar to AVANGO Lightning is also restricted to SGI platforms.

VRJuggler ([BJH$^+$01]) and *DIVERSE* ([KASK02]) are software systems for building large high-end VR applications. Both systems have been designed to overcome the drawbacks of some of the previous systems. VRJuggler establishes a modular architecture for different devices. To maximize performance the only layer the application directly interfaces to is the graphics API. Due to the complexity of VRJuggler the usage of its API incorporates much effort, whereas DIVERSE lacks platform independence since it is available for Linux and IRIS systems only. However, a Windows-based version is currently under development. Other approaches such as the Studierstube project ([SFH$^+$02]) and MRToolkit ([SGLS93]) have originally been developed for immersive VR systems only, but extend to other VR systems. However, these approaches provide insufficient support for projection-based VR environments.

A more detailed description of further VR software systems and toolkits can be found in [BJ98]. Although many systems are available for creating and developing VR-based applications, due to compatibility and customization issues universities and research institutions tend to use their own visualization libraries and VR-based extensions to explore and interact with their specific datasets. Thus there is no standardization of VR system architectures yet. In the next sections an alternative approach for a VR software system is presented. Since the system is based on a generic computer graphics system, this system is introduced first.

3.3 Virtual Rendering System

VRS, the *Virtual Rendering System* ([DH02]), is an object-oriented, scenegraph-based 3D computer graphics system written in C++. It provides numerous building blocks for composing 3D scenes, animated 3D objects, and the integration of 3D interaction. VRS concentrates on OpenGL for real-time rendering and supports advanced rendering techniques such as multipass algorithms for the simulation of global illumination phenomena. In addition, VRS wraps various other 3D rendering libraries, e.g., RenderMan, the global illumination system Radiance and also ray-tracing renderers such as POVRay. Since VRS completely encapsulates low-level 3D rendering systems through a uniform interface, applications are able to switch between different rendering systems without any need for reimplementing the application. In interior design prototyping, for instance, a modeled scene can be rendered for interactive exploration and sound propagation using OpenGL and an arbitrary spatial sound library. Rendering the scene with Radiance can simulate interior light dispersion, or a ray-traced image of this room can be generated using the POVRay interface.

Listing 3.1 shows the usage of different renderers for the same scene. A VRS *canvas*, which

```
1  ...
2  // root geometry node
3  SO<SceneThing> main = new SceneThing(view);
4  ...
5
6  #ifdef _RADIANCE_
7  SO<MExternCanvas> canvas = new MExternCanvas("Radiance-Win", 512, 512);
8  #endif
9  #ifdef _RENDERMAN_
10 SO<RMCanvas> canvas = new RMCanvas("RenderMan-Win", 512, 512);
11 #endif
12 ...
13 #ifdef _OPENGL_
14 SO<GLCanvas> canvas = new GLCanvas("OpenGL-Win", 512, 512);
15 #endif
16
17 // connect canvas and scenegraph
18 canvas->append(main);
19 ...
```

Listing 3.1: Using different rendering system in VRS.

represents the main drawing area, is generated with respect to the chosen renderer via precompiler directives. In contrast, applications based on other graphics systems require a reimplementation to switch to another rendering system. In line 7 and line 10 the canvas is initialized for the Radiance resp. the RenderMan port, while in line 14 the canvas is created for the OpenGL port.

VRS is available for most common platforms such as Windows, Linux, and Mac OS X. Major user interface toolkits such as Qt, GTK+, GLUT, Tcl/Tk and wxWidgets are supported. VRS serves as a framework and testbed for application-specific and experimental rendering and the design of innovative interaction techniques ([DH02]). For run-time debugging, developers can use the Tcl/Tk package *interactive VRS* (*iVRS*) ([KD02]), which is intended as a rapid prototyping graphics environment. iVRS provides access to almost all features of VRS through the scripting language Tcl/Tk without the need of compiling. Hence, applications can be debugged and modified without the need of recompiling, and thus the development process is accelerated.

3.3.1 Scenegraphs and Behavior Graphs in VRS

A VRS application requires at least one canvas, which is typically integrated into the application's user interface. The scene displayed in a canvas is defined by one or more *scenegraphs*, while interaction and animation is specified by *behavior graphs*. Thus the four essential components a VRS developer has to deal with are:

- a **canvas** representing a drawing area for VEs,

- hierarchical **scenegraphs** describing geometry and appearance of virtual objects,
- **behavior graphs** describing event-dependent and time-dependent behavior of virtual scenes, and
- **graphics** resp. **non-graphics objects** representing shapes, graphics and non-graphics attributes, which are evaluated when they are included in scenegraphs.

A scenegraph is a hierarchical data structure that organizes shapes, groups of shapes, and groups of groups that collectively define the content of a scene. Shapes and subtrees may be shared among multiple nodes, creating a *directed acyclic graph* (*DAG*). Scenegraphs are widely used to define complex 3D scenes consisting of a multiplicity of separate shapes. APIs, such as Open Inventor ([Gro05]), OpenGL Performer, and Java3D ([SRD00]), all support the creation and rendering of scenegraphs. Traditionally, scenegraphs contain shapes defined by surfaces, such as sets of polygons, but they are not restricted to polygonal representations, for instance, volume scenegraphs (VSG) containing voxel information are also possible ([Nad00]).

Modeling VEs with two types of graphs, scenegraphs and behavior graphs, is unique to VRS. During scene evaluation both the scenegraph as well as the behavior graph are traversed by the VRS engine as desribed in Section 3.3.2.

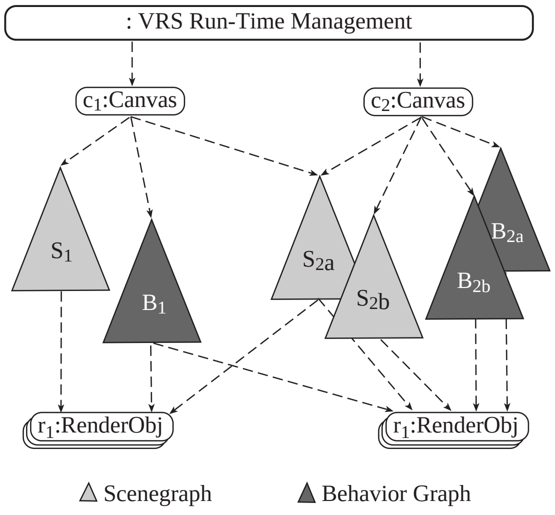


Figure 3.5: Concept diagram of a VRS application, containing scenegraphs and behavior graphs (adapted from [Döl05b]).

The relationships between canvases, scenegraphs, behavior graphs, and graphics objects are depicted in Figure 3.5. An application instantiates at least one canvas object, e.g., c_1 or c_2. A canvas can have more than one scenegraph, e.g., S_{2a} and S_{2b} for canvas c_2, and more than one behavior graph such as B_{2a} and B_{2b} for canvas c_2. Two or more canvas objects can share scenegraphs or behavior graphs as, for example, c_1 and c_2 share the scenegraph S_{2a}. All nodes in a scenegraph, so-called *scene nodes*, all behavior nodes, i.e., nodes in a behavior graph, and graphics objects are shareable objects, which can be referenced by clients multiple times. A behavior graph can also affect graphics objects referenced in scenegraphs or behavior graphs of other canvas objects such as B_1.

The behavior graph, which is used to define interactions by specifying animation or handling input events, consists of two types of nodes:

- *time-dependent nodes* can be used for the synchronization of animations, in particular time intervals can be defined in which certain actions are initiated, and
- *event-dependent nodes* can be used by the developer to generate corresponding system behavior as response to certain input events, e.g., a mouse click.

Using two types of graphs, scenegraphs and behavior graphs, is useful in the context of developing interactive applications. Because behavior is independent from geometry and most behavior cannot be assigned to a single subgraph, both aspects of an application should be treated independently. Canvas objects delegate events and requests, which are sent from the VRS run-time management to the associated graphs, e.g., redraw events to scenegraphs, and mouse or keyboard events to behavior graphs.

3.3.2 Scene Evaluation in VRS

In this section the evaluation of scenegraphs and behavior graphs in VRS is described.

Scenegraph Evaluation

The class `Engine` processes the graphics objects and attributes arranged in scenegraphs and delegates their evaluation to associated handlers. VRS scene content is defined by instantiating objects of the subclasses of `RenderObj`, which is inherited by all rendering objects of VRS. The rendering objects are defined by the classes `Attribute`, `Handler` and `Shape` and their corresponding subclasses (see Figure 3.6).

Objects of type `Shape` are representations of geometric shapes, e.g., `Box`, `Sphere` or `Torus`. Objects of the class `Attribute` represent graphics attributes applied during the scene evaluation. These graphics attributes are either of type `MonoAttribute` or of type `PolyAttribute`. During rendering only one `MonoAttribute` is active at any time. All objects of this type are managed using a stack, the top object is the active `MonoAttribute`. Examples of the type `MonoAttribute` are the `Camera` or the `Background` classes, but also classes associated with objects, such as textures or color attributes. In contrast, an arbitrary number of `PolyAttributes`, such as light sources or shadow caster objects, can be incorporated during scene evaluation.

The abstract class `Handler` provides the required service functionality to the engine and serves as the base class for all handlers used within VRS, it is inherited by several subclasses. The two derived classes `Simplifier` and `Painter` provide services to the engine to evaluate graphics objects as well as associated graphics attributes. Objects of the class `Simplifier` transfer VRS graphics objects into a representation that can be processed by the underlying rendering system, whereas objects of the class `Painter` are directly mapped to the underlying renderer.

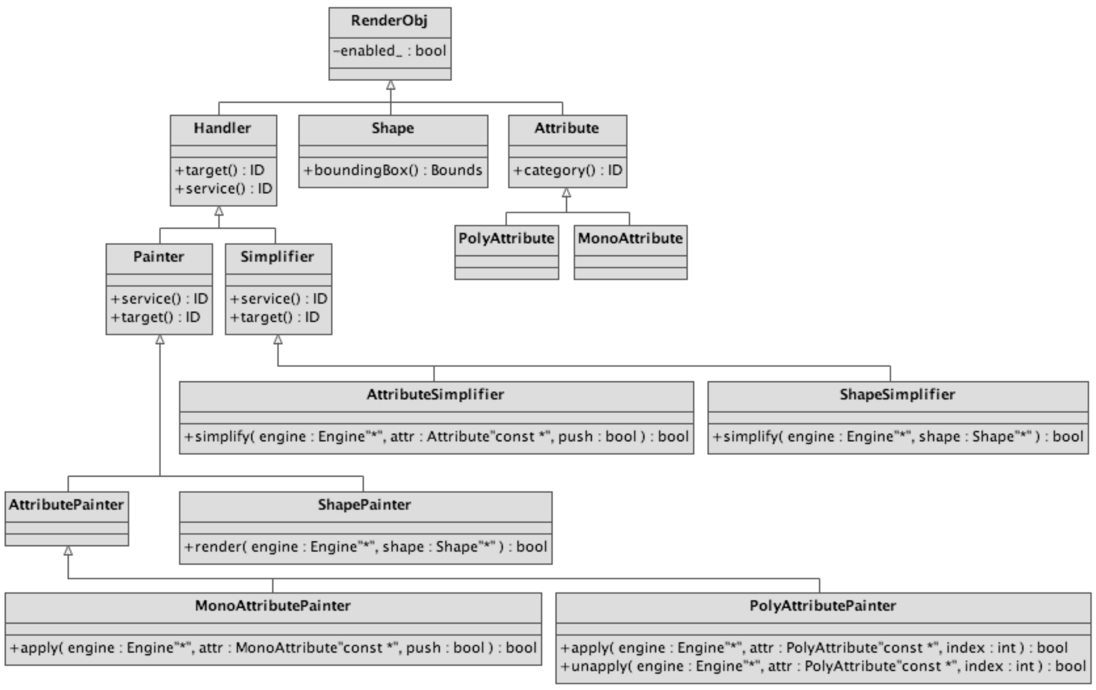

RenderObj
-enabled_ : bool
Handler
+target() : ID
+service() : ID
Shape
+boundingBox() : Bounds
Attribute
+category() : ID
PolyAttribute
MonoAttribute
Painter
+service() : ID
+target() : ID
Simplifier
+service() : ID
+target() : ID
AttributeSimplifier
+simplify(engine : Engine"*", attr : Attribute"const *", push : bool) : bool
ShapeSimplifier
+simplify(engine : Engine"*", shape : Shape"*") : bool
AttributePainter
ShapePainter
+render(engine : Engine"*", shape : Shape"*") : bool
MonoAttributePainter
+apply(engine : Engine"*", attr : MonoAttribute"const *", push : bool) : bool
PolyAttributePainter
+apply(engine : Engine"*", attr : PolyAttribute"const *", index : int) : bool
+unapply(engine : Engine"*", attr : PolyAttribute"const *", index : int) : bool

Figure 3.6: Classes for scene evaluation using VRS.

The VRS engine acts as a *status machine* to manage graphics contexts. Each graphics attribute can be changed independently without considering the entire context. The status of a context is defined by the contents of the different stacks, for example texture stacks or color stacks.

Although VRS enables the usage of different rendering systems, scene description is not limited to those features, which are common to all underlying systems. Rather VRS can support every feature of an arbitrary rendering system by introducing rendering system specific attributes and corresponding painters. Low-level code specific to a rendering system can be integrated into VRS source code; but this low-level code leads to a loss of flexibility of the rendering system.

Behavior Graph Evaluation

The behavior graph in VRS is of major importance, when specifying interaction for a virtual environment, which has been defined by a corresponding scenegraph. A user can specify dynamic behavior of a VE either manually, or it can be initiated synthetically. However, a system behavior requires an *event*, which causes a corresponding behavior. To support certain events, VRS provides the class `Event` and its two subclasses `TimeEvent` and `CanvasEvent` (see Figure 3.7). The former enables the processing of certain time events, which are initiated by the VRS clock. `CanvasEvents` are configure events, such as resize of a canvas, or several user input events, which are specified in the class `InputEvent`. Subclasses are, for instance, the class `ButtonEvent` to support processing of mouse button events, or the class `MotionEvent` to support handling of mouse motion events.

To enable an interaction process initiated by an input event, VRS provides the class `Condition`, which checks whether certain input events occur. For this purpose, the method `satisfied(Event* e)` is available, which returns `true` if the event `e` satisfies a certain condition and otherwise `false`. Depending on the return value the event can be further processed or ignored. A typical derived subclass is the `ButtonCondition` class, which checks whether specific button conditions are satisfied, e.g., the first mouse button is pressed once in a certain time interval without any keys pressed. An interaction process consists of three logical steps. It *starts*, it *executes* for a certain time interval, and it either *ends* regularly or is *canceled.* For a given event `e` and conditions `startCond_`, `processCond_`, `endCond_` and `cancelCond_`, the method `handle(Event* e)` calls

1. `start(e)`, if `startCond_` is satisfied,
2. `process(e)`, if `processCond_` is satisfied,
3. `end(e)`, if `endCond_` is satisfied, or `cancel(e)` if `cancelCond_` is satisfied.

Specific interaction tasks can be implemented in four methods `start`, `process`, `end`, and `cancel` by overloading these methods (see Figure 3.7). The conditions `startCond_`, `processCond_`, `endCond_`, and `cancelCond_`, that have to be satisfied, determine the state of the interaction. Each of these methods returns whether a redisplay of the canvas is necessary or not.

By using behavior graphs, leaves and subgraphs of a scenegraph can be modified with respect to their properties, such as position, geometry, color etc. In VRS a behavior itself is defined by instantiating objects of the class `BehaviorNode`, which represents all scene nodes containing behavior information. For this purpose, two different nodes are availbale: *leaf-behavior nodes* and *non-leaf-behavior nodes* (see Figure 3.7). Non-leaf behavior nodes coordinate dynamic procedures hierarchically and enable the developer to create complex animations as well as interaction. Non-leaf-behavior is defined by the classes `MonoBehavior` and `PolyBehavior`. Objects of the type `MonoBehavior` own exactly one child node, whereas objects of the type `PolyBehavior` reference an ordered set of child nodes. `MonoBehavior` and `PolyBehavior` pass the method `handle(Event* e)` to their child nodes, whereas `LeafBehavior` implements this method. The return type `BehaviorNode::InvalidationHint` defines to what extent a redisplay of the canvas is requiered. Possible redisplay directives can be `RedrawNothing`, i.e., nothing is redrawn, `RedrawWindow` which initiates a redraw of the content of the associated window, or `RedrawWorld` which renders the entire virtual world.

All classes specifying user interactions are derived from the class `LeafBehavior`, such as the `PickingCallback` or the `Interaction` classes. For example, with an instance of the type `PickingCallback` a callback can be executed when a specified shape is hit by the mouse pointer and given conditions are satisfied, i.e., mouse button, multiplicity, and additional modifier keys comply with the requirements of the condition. The argument passed to the callback is an object of type `IntersectionInfo` that is obtained by executing a ray query, which returns the shapes hit by the mouse pointer resp. a null object if no shape is hit.

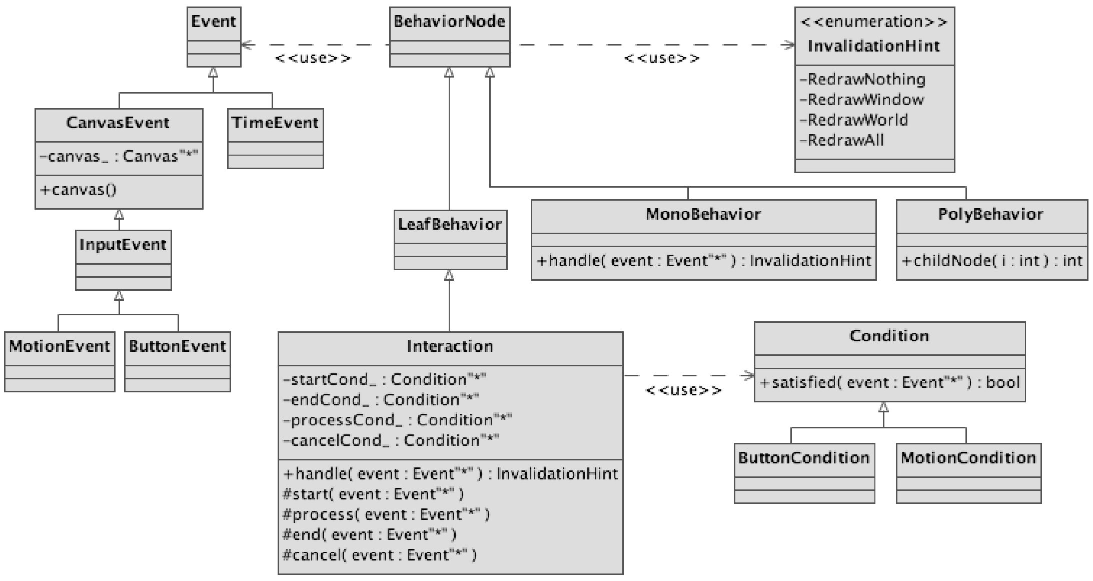


Figure 3.7: Classes used to realize interactions in VRS.

For the design of interaction strategies the class `Interaction` is essential. Examples of subclasses of the `Interaction` class are control classes for input devices; for instance objects of the class `Manipulator` receive and propagate mouse events. For six DoF manipulations the interaction is started, when the user performs a predefined action indicating the selection of a virtual object. During the six DoF manipulations the interaction executes until the user releases the object or an error occurs. An instance of type `Interaction` can receive and propagate canvas events, which belong to the interaction nodes appended to one of the behavior graphs of the corresponding canvas; other events are ignored. Further examples of classes derived from `Interaction` are discussed in Section 3.4.3.

3.3.3 Multipass-Rendering in VRS

As mentioned in Section 3.1.3, pipeline concepts enhance the performance when rendering virtual scenes. However, some rendering techniques require to traverse the scenegraph multiple times, because interactions between several objects have to be considered. For example, algorithms to simulate global illumination phenomena such as shadows or reflections require these additional scene traversals. For this purpose, VRS provides two different mechanisms: *multi-scene renderers* and *techniques*. When using multi-scene renderers multiple evaluation passes are supported, in which the entire scene description is processed. This is needed among others for *stereographic rendering*, in which the scene is rendered twice, i.e., once for each eye. The class `StereoRenderingGL` that handles these issues is described in Section 3.4.2.

Techniques are rendering strategies, which determine how the scenegraph has to be evaluated. In contrast to multi-scene renderers, techniques are capable of evaluating only subsets of the scenegraph. Since the scene needs not to be rendered entirely multiple times, techniques can be exploited to optimize performance. Processing of these multi-pass rendering algorithms within VRS is performed via the classes `Technique` and `TechniqueProcessor`. Certain techniques are activated by appending an instance of the corresponding subclass of `Technique` to the scenegraph; an instance of the class `TechniqueProcessor` attached to the scenegraph collects and processes theses techniques. In order to combine several techniques, each pass of technique has a *priority* that defines the sequence in which the techniques are processed. Table 3.1 illustrates the priorities for different rendering passes.

Each technique provides the following methods to handle the actions required to perform the corresponding technique. Initially, the `start` method of an active technique is executed to announce the necessary passes to the `TechniqueProcessor`. The method `activatePass` is called to submit the identifier of the next rendering pass, and the method `preparePass` is executed before each traversal. The method `finishPass` is processed after each scene traversal. By using this method settings of `preparePass` can be revoked. The rendering of each shape can be altered by the method `prepareEval`. This method is evaluated immediately before a shape is rendered or ignored. If this method returns `false` for only one technique, the shape is discarded in this pass, otherwise it is rendered accordingly. The method `finishEval` is used analogously to the method `finishPass` after the evaluation of shapes considered. Again, changes applied to the low-level rendering system before rendering a shape can be revoked in this method. The boolean return value of `canBeUsed` indicates whether requirements of a certain technique are fulfilled, e.g., requirements referring to the features provided by the graphics board.

Pass Name	**Pass Priority**
`BOUNDINGBOX`	5
`ACTIVEOBJECTTECHNIQUE`	10
...	
`MAIN`	100
...	
`TRANSPARENCY`	500

Table 3.1: Pass names and priorities for multiple-scene evaluation.

As mentioned before, passes are evaluated according to their priority. For example, the `BOUNDINGBOX` path ensures that the bounding box for each shape is accessible. During the `MAIN` pass each shape is rendered. After the scene evaluation is finished, all techniques are terminated by the `TechniqueProcessor` calling the `stop` method. Several multi-pass rendering techniques can be realized by these classes and methods, however, multiple rendering of shapes is not the only purpose of techniques. In a preevaluation phase of the scenegraph a technique of type `ActiveObjectTechnique` is used to calculate and store distances between virtual objects in the VE. The usage of this class is described in detail in Chapter 4. Multi-pass rendering techniques to generate virtual shadows and virtual reflections are considered in Chapter 5.

3.4 VR^{2}S: Virtual Reality Rendering System

In this section the extension of the high-level rendering system VRS to the generic VR software system *Virtual Reality Rendering System (VR^{2}S)*, sometimes referred to as *Virtual Reality VRS* ([SRH05a]), is discussed in detail. As described above, VRS provides flexibility in terms of the rendering system and the user interface toolkit. Thus rendering can be performed with several low-level rendering APIs such as OpenGL, RenderMan or ray-tracing systems, and the interface can be implemented by arbitrary user interface toolkits to support both desktop- and VR-based interaction. VR^{2}S meets the demands of VR developers as well as users and has demonstrated its potential in different planning and exploration applications. The distinction between geometry and its associated appearance on one side and its behavior on the other side, eases the development of interactive and immersive applications. Thus VR^{2}S enables rapid prototyping of interactive 3D applications.

In the following sections, the architecture of VR^{2}S is presented and essential concepts, e.g., stereoscopic rendering, integration of tracking systems, and the VR user interface are discussed in detail.

3.4.1 Software Architecture Overview

The main objective of VR^{2}S is to provide VR software developers with a suite of APIs that abstract and hence simplify all interface aspects of applications including the I/O interface and typical VR system tasks such as tracking. Thus developers can focus on interaction and system behavior. VR^{2}S supports interplay of highly independent modules by a multi-layered application programming interface. The platform independence as well as modularity of VR^{2}S eases both extensions as well as usability. The extensions of VRS by software components are based on the same paradigms as the entire design of VRS. VR^{2}S applications are independent of the VR system, and hence applications run on different system in both VR- and desktop-based environments.

As illustrated in Figure 3.8 VR^{2}S extends VRS by a VR-based user interface to support advanced multimodal HCI concepts. The VR-based user interface (see Section 3.4.4) provides sophisticated concepts for the development of interaction metaphors and techniques.

Since there are many libraries that support different interaction concepts, for example to interface multimodal input and output channels, these libraries can be used by mapping them to VR^{2}S. Besides libraries supporting multimodal interaction, e.g., auditory or haptic libraries, other external libraries are not focussed to support multisensory input or output channels and include further interaction concepts, such as physical simulation systems, e.g., *Open Dynamics Engine* (*ODE*) ([Smi04]), or the augmented reality software system *ARToolKit* ([KB99]) (see Chapter 5). In the graphics layer, some of the VR^{2}S extensions refer to the VRS core of rendering techniques, e.g., stereoscopic rendering, other components relate to the VR-based user interface at the application layer, e.g., interaction via six DoF input devices. As mentioned in Section 3.3, rendering of the scene is performed in the rendering layer by the usage of adapters to the corresponding rendering system. As illustrated in Figure 3.8 further systems contributing

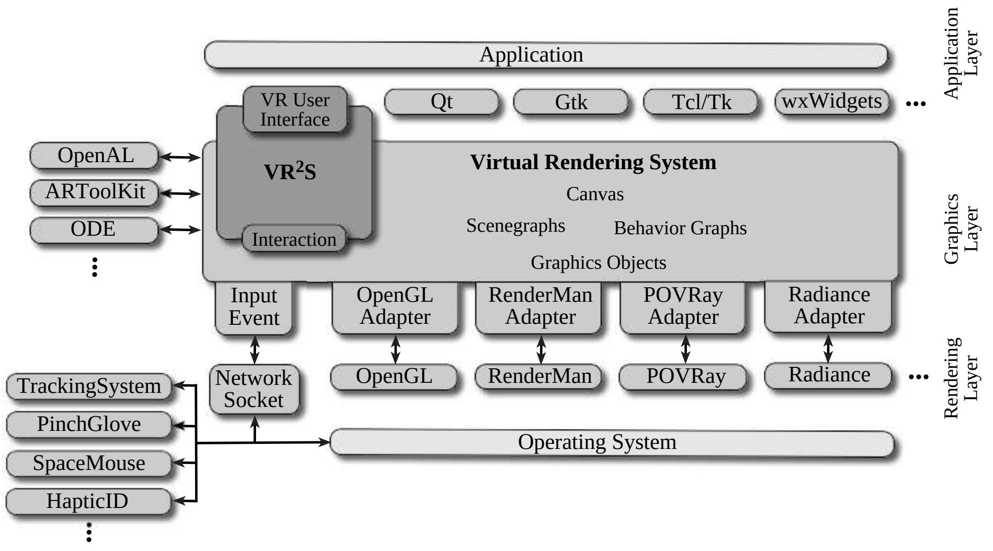


Figure 3.8: The system architecture of VRS ([DH02]) and its VR-based extension VR^{2}S consisting of application, graphics and rendering layers.

to multimodal interaction concepts, e.g., tracking system, pinch data glove system etc., can be connected to a VR^{2}S application either locally or via network sockets. The next section describes in detail a multi-scene renderer, which supports the rendering of stereoscopic images.

3.4.2 Stereographic Rendering

As pointed out in Chapter 2, stereoscopic viewing improves human's depth perception in VR. VR^{2}S provides an easy to use API to generate stereoscopic images. Whatever display and rendering system are used, for stereoscopic viewing two images, one for each eye, have to be be rendered each with its own camera settings. These settings are defined by an *orientation* and a *projection* matrix. The orientation matrix is obtained from an instance of the class `LookAt`, which is determined by three parameters *from*, *to* and *up*. *from* defines the position of the camera, *to* the focus point and *up* the camera's up-vector. The normalized vector r is defined as $r = \frac{dir \times up}{|dir \times up|}$, where the normalized direction vector *dir* is given by $dir = \frac{to - from}{|to - from|}$ (see Figure 3.10). Vector r points from the camera's position to the right orthogonally to the plane defined by the vector *dir* and the vector *up*. VRS provides two kinds of predefined projections, *orthogonal projections* and *perspective projections*. As described in Section 3.1.3 orthogonal projections preserve parallelism and are not suitable for stereoscopic images, because parallel projections conflict with perspective depth cues and therefore disturb depth perception. Hence perspective projections are used for stereographic rendering in VR^{2}S. An Instance of the class `Perspective` generates the projection matrix, which is defined by the *near*- and *far*-clipping planes, the camera aperture in the *up*-direction *fovy* (field of view in y-direction) and the ratio of the view plane; these parameters together define the *view frustum*. In addition, for stereographic rendering the user also has to define the previously mentioned interpupilar eye distance *IPD*, which determines the camera

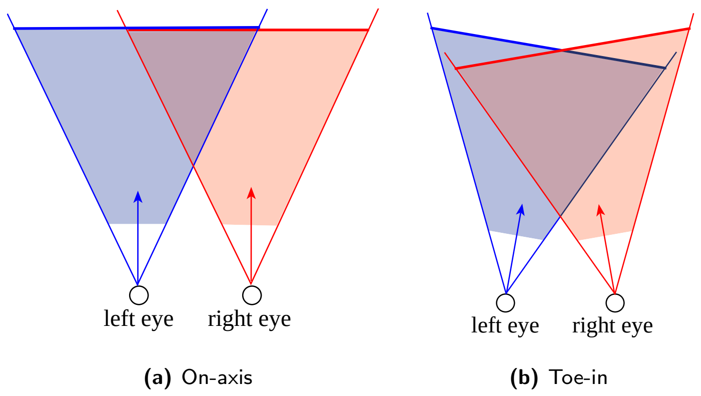


Figure 3.9: Stereographic rendering techniques illustrating the on-axis technique (a) and the toe-in technique (b).

shift between both eyes.

There are different stereographic rendering techniques to produce stereoscopic images. All stereographic rendering approaches have in common that two images have to be rendered, one for each eye. This is achieved by translating the camera before rendering by the half of the interpupiliary distance to the left resp. to the right along the aforementioned vector r, i.e., $-\frac{IPD}{2}$ for the left eye and $+\frac{IPD}{2}$ for the right eye. Although this so-called *on-axis* approach ([Bou99, HM85]) provides stereoscopic images without any further steps, it has the following drawback. Because of the translation the image planes for both images diverge, and the binocular region in which a stereo image is perceivable smaller than the original view frustum (see Figure 3.9 (a)). Other approaches address this shortcoming and support an enlarged stereoscopic region. Most of these stereographic rendering approaches are incorrect since they introduce *vertical parallax*, i.e., a vertical mismatch between points in both images. For example, the *toe-in method* ([Bou99, Sou92]) involves a rotation of the scene such that the image planes for both eyes intersect. The camera is rotated inwards by the angle $\pm\arctan(\frac{0.5 \cdot IPD}{|to-from|})$ such that the view planes cross in the focus point (see Figure 3.9 (b)). Similar methods are still often used although they only approximate stereoscopic images, because the correct *off-axis method* ([Bou99]) requires features that are not supported by all rendering packages. However, since VRS supports access to the underlying low-level APIs, VR^{2}S implements the correct off-axis approach for OpenGL.

Figure 3.10 illustrates this approach. The view frustums for both eyes are sheared in such a way that their projection planes become identical. Therefore, the standard projection matrix P, specified by the application developer, is multiplied by a matrix M, such that $P \cdot M$ generates an asymmetric view frustum when used in the rendering pipeline. The matrix M is defined as follows:

$$M = \begin{pmatrix} \frac{2 \cdot near}{right-left} & 0 & \frac{left+right}{left-right} & 0 \\ 0 & \frac{2 \cdot near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{far+near}{far-near} & -\frac{2 \cdot far \cdot near}{far-near} \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

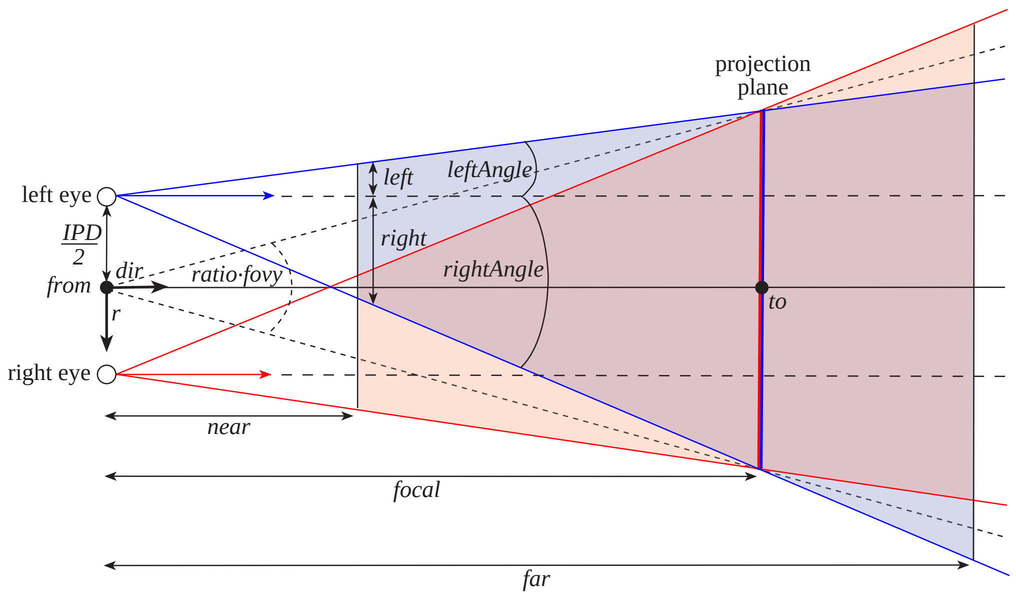


Figure 3.10: Illustration of the off-axis stereographic rendering technique (adapted from [Bou99]).

In the case for the left eye, the values of this frustum matrix are defined by

$$\begin{aligned} left &= -\tan(leftAngle)\cdot near\cdot ratio + \frac{IPD\cdot near}{2\cdot focal} && (3.1)\\ right &= -\tan(rightAngle)\cdot near\cdot ratio + \frac{IPD\cdot near}{2\cdot focal} && (3.2)\\ top &= \tan(topAngle)\cdot near && (3.3)\\ bottom &= \tan(bottomAngle)\cdot near && (3.4)\end{aligned}$$

For the right eye equations (3.1) and (3.2) have to be replaced by

$$\begin{aligned} left &= -\tan(leftAngle)\cdot near\cdot ratio - \frac{IPD\cdot near}{2\cdot focal} && (3.5)\\ right &= -\tan(rightAngle)\cdot near\cdot ratio - \frac{IPD\cdot near}{2\cdot focal}, && (3.6)\end{aligned}$$

equation (3.3) and (3.4) for *top* and *bottom* remain unchanged for the right eye. The values for the parameters *left*, *right*, *top*, *bottom*, *front*, *back*, *leftAngle*, *rightAngle*, *topAngle* and *bottomAngle* (see Figure 3.10) are derived from the camera settings stored in the `CameraInfo` class. The *focal length* given by *focal* is defined as the orthogonal distance from the camera position *from* to the focus point *to* on the projection plane (see Figure 3.10). Objects which are closer to the camera than the focal length have a *negative parallax*, and objects which are farther away have a *positive parallax*. When viewing stereoscopically, objects with a negative parallax appear to be in front and those with a positive parallax behind the projection screen. Objects with *zero parallax* appear to be located on the projection screen like two-dimensional flat images.

As mentioned before, not all graphics systems support the generation of asymmetric view frustums. OpenGL provides a function call: `glFrustum(left,right,near,far,top,bottom)`

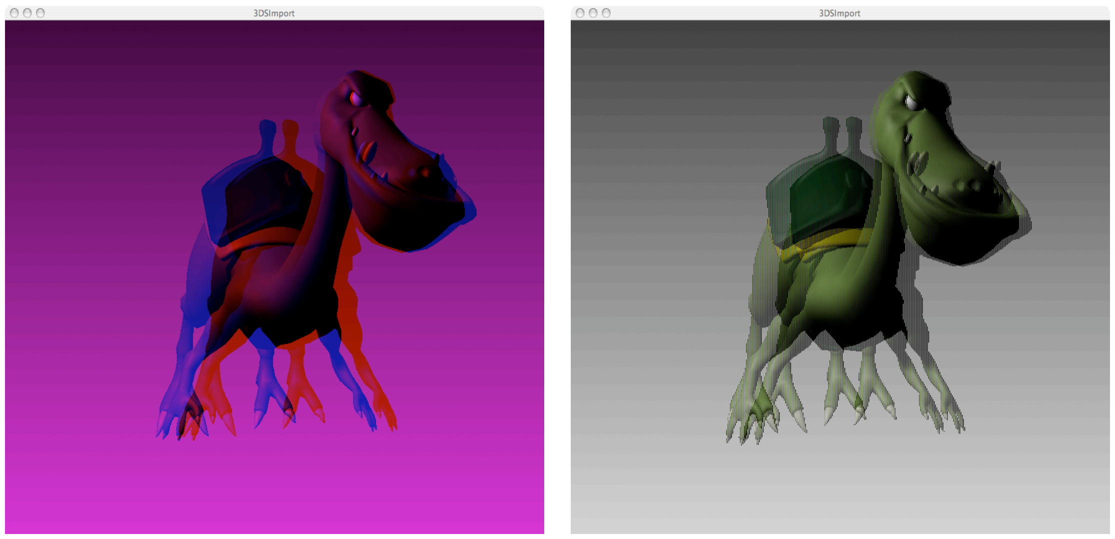
```
1  ...
2  SO<GlutCanvas> canvas;
3  canvas = new GlutCanvas("GlutStereo", 400, 300,
4                          GLCanvas::RGBDDS || GLCanvas::STEREOVIEWING);
5
6  SO<StereoRendererGL> stereoGL;
7  if (canvas->hasProperty(GLCanvas::STEREO_VIEWING)) {// quad buffer mode
8     stereoGL = new StereoRendererGL(StereoRendererGL::PAGEGLIPPING,0.3);
9  } else {// interlaced mode
10    stereoGL = new StereoRendererGL(StereoRendererGL::INTERLACE,0.3);
11 }
12 canvas->setSceneRenderer(stereoGL);
13 ...
```

Listing 3.2: Stereographic rendering in VR^{2}S.

for this purpose. But the matrix generated by this function differs from the required matrix M since OpenGL uses a right-handed coordinate system, whereas the coordinate system used by VRS is left-handed.

To achieve stereographic rendering in VRS the canvas multi-scene renderer is set to stereographic rendering, i.e., the canvas multi-scene renderer is assigned an instance of the class `StereoRendererGL`. When using this multi-scene renderer, the scene is evaluated twice, with the corresponding camera settings as described above. Listing 3.2 illustrates the usage of this class. In lines 2 to 4 the frame buffer with the required properties is initialized, i.e., OpenGL specific quad-buffer stereo providing four frame buffers for eye-dependent double buffering (see Section 3.1.3) and support for the stencil buffer. In line 7 an object of type `StereoRendererGL` is defined for time-sequential stereo, if quad-buffer stereo is supported by the graphics card, whereas in line 10 stereographic rendering is defined for spatially sequenced stereo images for autostereoscopic displays using the stencil buffer.

In Figure 3.11 a simple VRS scene containing a *3DS* (*3D Studio Max format*) model of a dinosaur is rendered using the `StereoRendererGL` class. In Figure 3.11 (a) the scene is rendered in the `REDBLUE` anaglyph stereographic mode, i.e., the image for the left eye is rendered with a red color mask, while the image for the right eye is rendered with a blue color mask. The stereoscopic effect can be viewed with properly colored anaglyph glasses. Figure 3.11 (b) illustrates the interlaced stereographic mode, i.e., the image for the left eye is rendered in the even columns, while the image for the right eye is rendered in the odd columns. This mode is suitable for autostereoscopic displays, which provide a corresponding *lenticular* or *barrier mask* in front of the screen to separate the two images.



(a) Anaglyph stereoscopic image **(b)** Interlaced stereoscopic image

Figure 3.11: A simple VRS scene containing a 3DS model of a dinosaur rendered (a) in an anaglyph stereographic mode and (b) in interlaced stereographic mode.

3.4.3 Tracking Systems in VR^{2}S

As mentioned in Chapter 2, motion parallax is an important depth cue that denotes the fact that when the objects or the viewer move, objects which are farther away from the viewer seem to move slower than objects closer to the viewer. Although stereoscopic images provide an essential depth cue, motion parallax cannot be support without any further effort. To reproduce this effect in VR, *head tracking* and *view-dependent rendering* is required. For view-dependent rendering the position and orientation of the user's head need to be determined to map this pose to the virtual camera defined in the VE. Tracking systems allow to track certain physical objects, e.g., the pose of a user's head, by exploiting different magnetic, ultrasonic or optical technologies. The tracked devices can be associated with the user's head, but also with input devices for VR-based interaction.

Since high accuracy and wireless interaction are essential for precise and comfortable VR-based interaction, *optical tracking systems* appear to be most suitable to determine position and orientation of arbitrary objects. In contrast to mechanical approaches no wires disturb the interaction. Furthermore interferences that may occur when using magnetic or ultrasonic technology are prevented by the usage of optical tracking systems. For stereo-based optical tracking systems accuracy is in the range of submillimeters, and thus tracking errors are minor and precise interactions with objects displayed in the VE are possible.

Since lighting in VR system environments usually has to be dimmed significantly because the brightness of the projection screens, e.g., CAVEs, is limited, most often *infrared (IR)* light in combination with infrared pass filters attached to the cameras' lenses is used. IR optical tracking systems measure positions of numerous *markers* in a certain range and return the results with a time delay of 20 to 40ms, which is referred to as *real-time tracking* ([DU02]). A *target*, sometimes denoted as *rigid body*, is a fixed geometrical arrangement of at least three markers. Because of the many drawbacks of *light-emitting diodes (LEDs)*, usually passive markers made of simple objects, e.g., spheres, and wrapped into a retro-reflective material are used. The IR cameras illuminate the working area with an IR flash invisible to the user, and only the markers reflect the infrared light back to the cameras. Only this light reflected by the markers passes the infrared pass filters which are attached to the lenses. Grabbed frames, i.e., two-dimensional images taken by the cameras, consist of white pixels representing markers and black pixels elsewhere. When a marker is detected by at least two cameras, the position in the 3D space can be calculated by reprojection ([DU02]). The cameras have to be calibrated, i.e., their relative positions and orientations have to be determined, and their absolute positions and orientations have to be defined in terms of a user-defined tracking coordinate system. This is usually done by a *room calibration set*, consisting of a wand, i.e., a stick with two markers, and an angle carrying four markers to define the tracking coordinate system. The distance between the two markers of the wand has to be registered by the tracking system. The wand is then used to take a sufficient set of measurements from which the tracking system can calculate the correlation between the cameras as described in [DU02]. Finally, in order to define the origin and orientation of the tracking coordinate system, the L-shaped angle is positioned in the working area tracked by the cameras. Usually this tracking coordinate system is adapted to the position and orientation of the projection screen of the display device used in combination with the tracking system (see Figure 3.13 (left)).

When targets are detected by the tracking system the pose data for each target consist of Cartesian coordinates x, y, z defining the position of the target and three angles r_x, r_y, r_z that each describe the rotation of a target around the corresponding Cartesian coordinate axis giving the orientation of the target. To enable determination of these values all targets have to be registered by the tracking system. For this purpose, users have to define a *reference target* for each trackable rigid body. The relation between the current pose of the tracked target and its reference definition specifies the *rigid body transformation* of the target. This rigid body transformation is applied to a virtual object which is associated with a corresponding target, i.e., the determined transformation matrix representing the pose of the target is applied to the associated scene object.

The reference target can be defined in the *biovision human format* (BVH) ([PHH$^+$03]). The BVH-file format is a joint-based format, which also supports *motion tracking* approaches to grab and store user movements. A BVH-file has two parts, a header section that describes the hierarchy and initial pose of the rigid body, and a data section, which contains the motion data. For example, the header part of a BVH-file for the rigid body attached to the input device illustrated in Figure 3.12 is given by Listing 3.3.

In line 1 the keyword `ROOT` followed by the name of the segment starts the hierarchy to be defined. The vector after the keyword `OFFSET` specifies the offset of the current segment to its parent; in this case the offset is a zero vector. The `JOINT` definition in Line 3 defines the position of the first joint. The keyword `EndSite` indicates that the current joint ends at the given offset.

```
ROOT InputDevice {
    OFFSET 0.0 0.0 0.0 //m1
    JOINT m2 {
        OFFSET −9.0 0.0 −170.5 //m2
    }
    EndSite {
        OFFSET −80.0 0.0 −184.0 //m3
   }
}
```

Listing 3.3: Rigid body BVH-file.

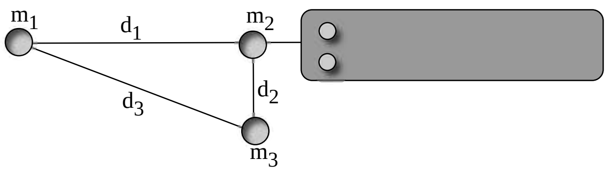


Figure 3.12: An example rigid body arrangement at the top of an input device.

Hence, the BVH-file defines a target illustrated in Figure 3.12 consisting of three points $m_1 := (0,0,0)$, associated with the marker at the top of the input device, point $m_2 := (-9.0, 0.0, -170.5)$ associated with the marker in front of the handle, and $m_3 := (-80, 0.0, -184.0)$ associated with the marker at the top of the stick branched to the left. When tracking such a target from a set of predefined rigid bodies, the tracking algorithm scans the point cloud resulting from numerous markers for the distances d_1 between m_1 and m_2, d_2 between m_2 and m_3, and d_3 between m_1 and m_3 (see Figure 3.12). If these distances are detected the position and orientation of this target with respect to its reference definition can be determined by a rigid body transformation as follows.

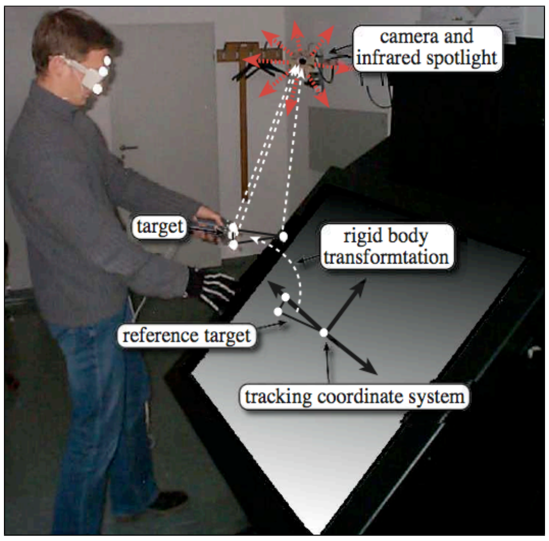
Let P_0, P_1, P_2 denote the tracked position of three markers of a target. We define the vectors

$$\begin{aligned} R_z &:= \frac{P_1 - P_0}{|P_1 - P_0|} \\ R_x &:= \frac{(P_2 - P_0) \times (P_1 - P_0)}{|\,(P_2 - P_0) \times (P_1 - P_0)\,|} \\ R_y &:= R_z \times R_x, \end{aligned}$$

and the transformation matrix

$$T = \begin{pmatrix} R_x[0] & R_x[1] & R_x[2] & -P_0[0] \\ R_y[0] & R_y[1] & R_y[2] & -P_0[1] \\ R_z[0] & R_z[1] & R_z[2] & -P_0[2] \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where $R[i]$ denotes the i-th component of the vector R. Applying this matrix T to the target



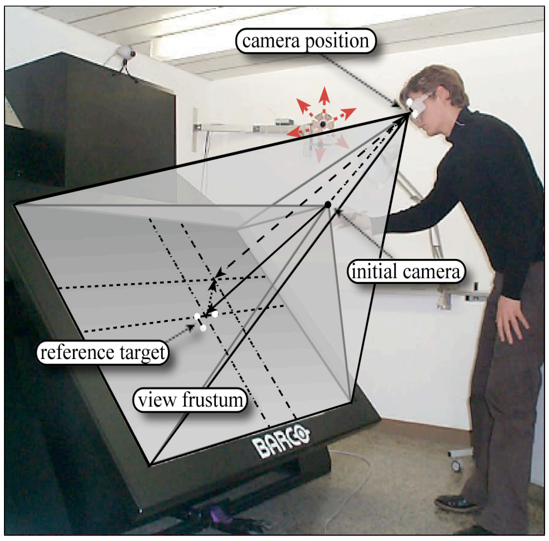


Figure 3.13: An optical tracking system and a corresponding rigid body transformation (left), and head tracking for view-dependent rendering (right).

defined by three tracked points P_0, P_1 and P_2 transforms P_0 to the origin, P_1 onto the positive z-axis and P_2 into the y-z-plane. Let O denote the matrix that transforms the reference target to the target transformed by T. Hence, the matrix

$$T_{RB} = T^{-1} \cdot O$$

determines the rigid body transformation that is applied to the virtual object associated with the target. T_{RB} transforms the virtual object associated with the reference definition of the target with respect to the currently tracked pose of the target.

Figure 3.13 (left) illustrates such rigid body tracking and transformation. The user is equipped with an input device that is attached with passive markers. IR diodes arranged on a ring around the camera illuminate the scene. The target associated with the input devices reflects the infrared light and the tracking system reconstructs the position and orientation in the 3D space with respect to the coordinate system as described above.

When several trackable devices moves in the working area of an optical tracking system, the tracking system calculates the position and orientation of each of these devices and generates a data package, consisting of a unified identifier of the target as well as a (4×4)-matrix representing the rigid body transformation of the target. The application developer assigns a target to a physical input device as well as to a virtual input device. When tracking the physical input device, i.e., the associated target is detected, the resulting pose data is applied to the virtual input device associated with the target.

Since the tracking process requires much effort in terms of processor resources and memory, the tracking system usually runs on a different machine than the graphics system, and communication is performed via network connection as illustrated in Figure 3.8.

```
1  ...
2  // initialization of the tracking system
3  SO<TrackingSystem> ts = new TrackingSystem(camera);
4
5  // set ID of the tracked glasses
6  ts->setHeadID(0);
7
8  // append tracking system to the canvas
9  canvas->append(ts);
10 ...
```

Listing 3.4: Integration of tracking systems in VR^{2}S.

VR^{2}S provides the class `TrackingSystem` derived from `Interaction` which processes a data package generated by the tracking system. This class receives the data packages and generates corresponding event objects of type `TrackingEvent`. These objects contain attributes such as the rigid body transformation matrix and an identifier that associates the tracking event to a certain device. Tracking conditions can be defined by objects of type `TrackingCondition`, e.g., to ignore particular targets. If a virtual input device is connected to a target, which is recognized by the tracking system, the rigid body transformation is applied to the virtual input device and the visual representation of this device, e.g., a virtual hand, is placed accordingly.

As discussed in the beginning of this section, besides the control of virtual input devices tracking systems enable the simulation of motion parallax by using view-dependent rendering. Therefore, a target has to be associated with the head position of the user, e.g., by attaching it to the stereo glasses. When the tracking system is aware of this position and orientation of the user's head, the virtual camera is positioned and oriented in the virtual scene accordingly. To enable realistic motion parallax it is important that the corresponding view frustum extends to the corners of the projection screen at any time. In Figure 3.13 (right) this approach is illustrated. A target is attached to the stereo glasses of the user, and the transformation from the reference target to the current pose is used to define the view frustum with respect to the initial reference definition of the virtual camera.

Listing 3.4 shows how such a viewpoint-dependent rendering is activated in VR^{2}S. In line 3 an object of type `TrackingSystem` is created to determine the user's head position. The constructor's only parameter is the camera attribute, which is modified when viewpoint-dependent rendering is active. Other optional parameters include, for instance, the dimensions and orientation of the projection screen or additional transformations applied to the tracking data. In line 6 the (default) ID is assigned for head tracking, i.e., the pose of a device attached with a target, which is registered at the tracking system with the ID number 0, is used for the position and orientation of the virtual camera. Finally, in line 9 the tracking system object is connected to the canvas to evaluate all tracking events by this canvas.

3.4.4 VR User Interface

Besides for viewpoint-dependent stereographic rendering the passive IR-based optical tracking system can be used for tracking arbitrary devices with six DoF and exploiting them, for example, as input devices. Furthermore special VR hardware such as data gloves can be involved in a VR interaction process. To use such devices in VR^{2}S the class `SpatialInputDevice` provides an uniform interface for accessing virtual input devices. A virtual input device can be controlled via sampled data received from arbitrary hardware and abstracts from the particular used input device. Special VR hardware devices are either connected to the same computer VR^{2}S runs on or they can communicate via network sockets with VR^{2}S. Thus processing intensive and also platform dependent processes such as the described tracking or gesture recognition can be distributed to different systems. When an arbitrary VR input device is used, corresponding events of type `SpatialInputDeviceEvent` are generated and processed by the interaction handling mechanism. The events for a virtual input device are divided into four groups:

1. **Motion events** are generated and processed when an associated input device is tracked while moving through the VR system environment.

2. **Selection events** occur when the user performs a predefined action associated with the selection process, e.g., pressing a button or pinching fingers with a data glove.

3. After the user has initiated a selection process, **manipulation events** are generated permitting six DoF manipulations of virtual objects.

4. Analogous to the selection subprocess, **release events** are generated by the user performing predefined release actions, e.g., releasing a button.

The resulting events cause corresponding system behavior. A selection event causes grabbing, a release event causes releasing of virtual objects, and if a manipulation or motion event occurs, the virtual input device resp. selected objects are positioned in the scene according to the pose of the physical input device. Such interaction processes can be implemented by a callback mechanism, i.e., motion events, selection events, manipulation events and release events lead to execution of user-definable callbacks in which all concepts are handled. Listing 3.5 describes how an arbitrary scenegraph element can be used as a virtual input device.

In lines 2 to 4 a red-colored virtual sphere is appended to an object of type `SceneThing`, which represents a collection of graphics objects. In line 7 the scene node containing the sphere is appended as a virtual input device and can be controlled by arbitrary physical input devices, such as tracked data gloves. Since no callbacks are passed to the constructor, the virtual sphere can only be moved via an arbitrary physical input device and no selection or manipulation can be performed in this case. Since the `SpatialInputDevice` class incorporates further information, several interaction metaphors can be implemented by inheriting the callbacks (see Chapter 4).

VR input devices not already supported by VR^{2}S can be integrated into VR^{2}S to enable control via the described VR user interface. Besides desktop devices such as mouse or keyboard VR^{2}S supports data gloves, space-mouse and arbitrary tracked input devices. Since any physical

```
...
SO<SceneThing> thing = new SceneThing(view);
thing->append(new ColorAttribute(Color(1.0,0.0,0.0,1.0)));
thing->append(new Sphere());

// append spatial input device to the canvas
canvas->append(new SpatialInputDevice(thing));
...
```

Listing 3.5: A spatial input device in VR^{2}S.

object can be used as a VR input device by applying appropriate tracking technologies. Also user-defined input devices such as wands, gloves or hands can be used for intuitive six DoF interactions.

Since VR hardware devices are expensive, VR^{2}S is designed in such a way that all devices can be simulated. This allows the implementation and testing of VR interaction concepts outside a VR system environment. For instance, head tracking events can be simulated using the arrow keys on the keyboard.

3.4.5 Integration of Multisensory Feedback in VR^{2}S

In this section the integration of multimodal interaction concepts, in particular auditory and haptic feedback, in VR^{2}S is described. Both feedback types are usable in desktop-based and VR-based environments. In Chapters 4 and 5 multimodal interaction metaphors are presented that use these implementations to generate multimodal feedback during an interaction. This feedback can be used to give additional multisensory information, for example, about collisions or *snapping*, sometimes referred to as docking, of virtual objects.

Integration of Auditory Feedback

The integration of auditory feedback has been realized by mapping the features provided by the *OpenAL API* ([Hie05]). OpenAL is a cross-platform spatial audio library that models a collection of audio sources moving in three-dimensional environments that are heard by a single listener somewhere in the VE. The basic OpenAL objects are a *listener*, a *source*, and a *buffer*, which contains audio data. Each buffer can be attached to one or more sound sources, which represent points in the VE that emit audio. Besides by their 3D position sound sources can be characterized by a sound level, distance gain, or a conic propagation direction. One listener object has to be specified per audio context. This listener object represents the position where the sources are heard and is therefore connected to the virtual camera. With respect to the properties of the sound sources and the current virtual listener's position and orientation, OpenAL synthesizes the spatial sound. As described in Section 3.1.3, through fading when the distance between the sound source and the virtual camera increases the user gets the impression of spatial sound. In addition the *environmental audio extensions* (*EAX*) developed by Creative Labs enable the simulation of sound effects caused by the surrounding. Thus special sound effects caused by arbitrary environments can be reproduced, for example, sound effects caused by a hall or by a cave.

To support this functionality provided by OpenAL in VR^{2}S, sound sources, buffers and the listener are wrapped by certain wrapper classes. Before spatial sounds can be played in a VR-based application, the OpenAL sound renderer needs to be initialized to provide access to the sound devices and a sound context. The sound context handles the state of OpenAL similar to a graphics context, e.g., the graphics context of OpenGL. This initialization is performed by instantiating an object of the class `OpenALInitializer` (see Figure 3.14). Sound sources and listeners are represented by the classes `SoundSourceAL` and `SoundListenerAL` appended to scene nodes in the scenegraph. Sources and the listener can be fix, or their positions and orientations are implicitly given by their associated scene nodes. When associating a virtual listener with the virtual camera, the listener's position is updated according to the movement of the virtual camera. Thus, the perception of the sound corresponds to the cognition of the visual representation of the sound source. For example, the sound generated by a collision between objects propagates from the position where the user has seen the collision. Since there is only one listener per context, an object of type `SoundListenerAL` is derived from the class `MonoAttribute`. The different attributes of sound sources as well as listeners are described in detail in [Hie05, Rol05]. The constructor of the class `SoundBufferAL` provides access to a file containing the desired sound source, which has to be available in the *waveform audio format (WAV)*. Sound buffers can be associated with a sound source by passing the buffer address returned by `getBufferAddressAL` to the corresponding sound source.

Since the class `SoundSourceAL` is derived from the class `Shape` the evaluation of the sound sources can be realized by an instance of type `SoundSourcePainterAL` using the painter mechanism of VRS (see Section 3.3.2). An object of this type assures that playing of sounds is started when required and that the properties are set correctly. Therefore, the painter ensures that a sound source is only started, if it is not already playing. To set up the position correctly, the initial position of the sound source is multiplied by the current model matrix of the scene node that is associated with the sound source. If the listener is associated to a scene node the calculation of the position and orientation of the listener is performed analogously. If the listener is associated to the virtual camera, the initial state matrix of this listener is multiplied by the current view matrix and thus transforms the position and orientation of the listener according to the position and orientation of the virtual camera.

Integration of Haptic Feedback

In contrast to the integration of sound, haptic feedback is not based on a special haptic library. The implementation is realized by using a callback mechanism which allows to generate haptic feedback with respect to the features of the input device.

An example device supporting haptic feedback is illustrated in Figure 3.15. This wand-based input device has been constructed in cooperation with the technical services of the Westfälische Wilhelms-Universität Münster. It is equipped with a vibration unit and two input buttons. In

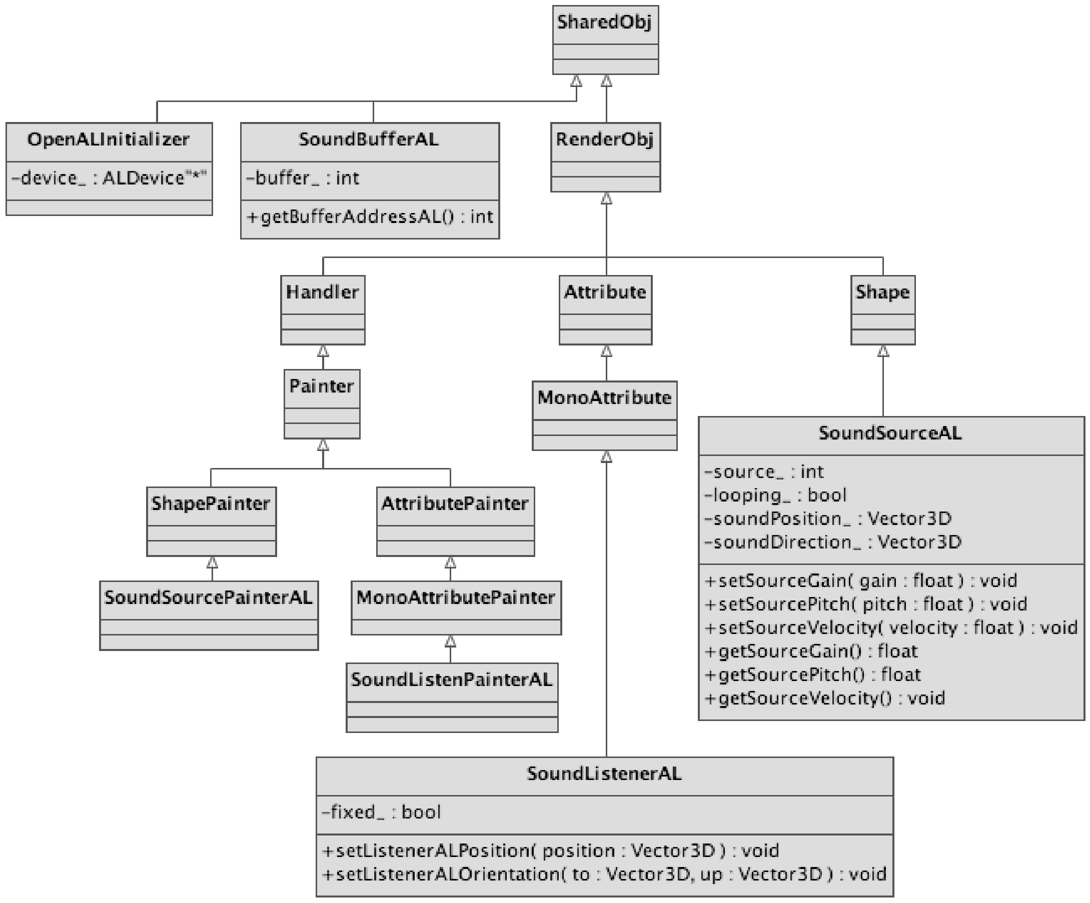

SharedObj
OpenALInitializer
-device_ : ALDevice"*"
SoundBufferAL
-buffer_ : int
+getBufferAddressAL() : int
RenderObj
Handler
Attribute
Shape
Painter
MonoAttribute
ShapePainter
AttributePainter
SoundSourcePainterAL
MonoAttributePainter
SoundListenPainterAL
SoundSourceAL
-source_ : int
-looping_ : bool
-soundPosition_ : Vector3D
-soundDirection_ : Vector3D
+setSourceGain(gain : float) : void
+setSourcePitch(pitch : float) : void
+setSourceVelocity(velocity : float) : void
+getSourceGain() : float
+getSourcePitch() : float
+getSourceVelocity() : void
SoundListenerAL
-fixed_ : bool
+setListenerALPosition(position : Vector3D) : void
+setListenerALOrientation(to : Vector3D, up : Vector3D) : void

Figure 3.14: Classes used to integrate OpenAL functionality into VR^{2}S.

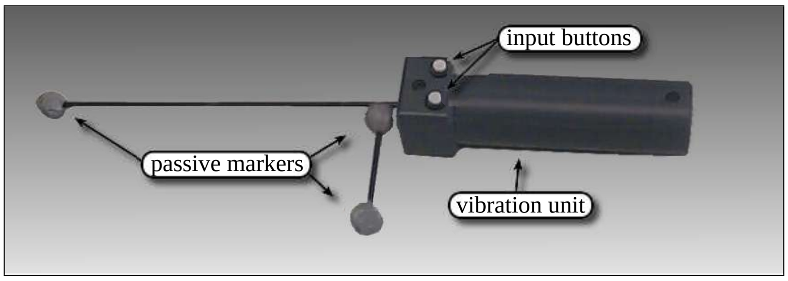


Figure 3.15: The prototype of the haptic input device.

addition, three passive markers for six DoF tracking are attached to the device as described in Section 3.4.3. To maintain the unconfined freedom of movement a wireless connection is used for transmitting vibration level information and button events. The vibration unit is connected to a bluetooth module that can be attached to the belt of the user. This module exchanges signals wirelessly with a bluetooth adapter connected to the serial port of the host computer.

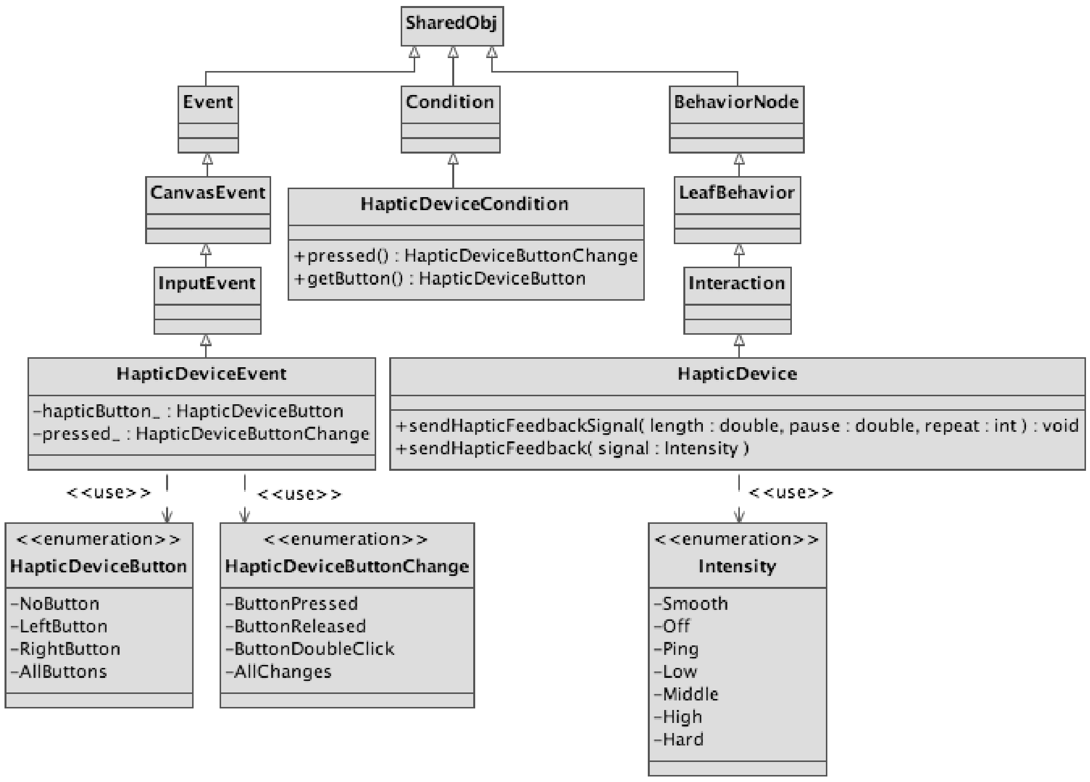


Figure 3.16: Classes used to integrate haptic feedback into VR^{2}S.

Before using the haptic input device, it has to be registered in the VR application by appending an instance of the class `HapticDevice` to the scenegraph that specifies the application. The class `HapticDevice` is derived from the class `Interaction` and opens the port to which the device is connected (see Figure 3.16). Via this port the developer can send one of several predefined vibration signals, e.g., smooth, medium and hard signals, with the method `sendHapticFeedbackSignal`; the parameter of this method defines the vibration level. In addition, further signals can be defined by a sequence of signals and breaks with varying intervals.

The two buttons allow the user to generate input signals, which can be interpreted as input events, for example, for selection tasks or contextual menus. Therefore, instances of the class `HapticDeviceEvent`, which is derived from the class `InputEvent`, are generated when a button is pressed. As described in Section 3.3.2 these events are further processed by the corresponding behavior graph. Furthermore conditions with respect to input events of the haptic device can be defined by objects of the class `HapticDeviceCondition`. This class can be used, for example, to combine events caused by a haptic input device and a pinch data glove in order to support *two-handed interaction*, sometimes referred to as *both-handed interaction* via different input devices ([BM86]).

```
#include <vrs/vrs.h>
#include <vr2s/vr2s.h>

int main (int argc, char** argv) {
    glutInit(&argc, argv);

    // initializing acoustic
    SO<OpenALInitializer> initAL = new OpenALInitializer();
    SO<SoundBufferAL> bufferAL = new SoundBufferAL("example.wav");
    SO<SoundListenerAL> listenerAL = new SoundListenerAL();

    // root node
    <SceneThing> myRoot= new SceneThing();
    SO<Camera> camera;
    camera = new Camera(Vector(0, 0, 5), Vector(0, 0, 0), 60)
    myRoot->append(camera);

    // virtual input device with sound
    SO<SceneThing> sid = new SceneThing(myRoot);
    sid->append(new Sphere());
    sid->append(new SoundSourceAL(bufferAL->getBufferAddressAL()));

    // building scene and behavior graphs
    SO<GlutCanvas> canvas = new GlutCanvas("Glut-VR2S", 1024, 768);
    canvas->append(myRoot);
    canvas->append(new SpatialInputDevice(camera, sid));
    canvas->append(new TrackingSystem(camera));
    canvas->append(new HapticDevice()));

    glutMainLoop();
    return 0;
}
```

Listing 3.6: Example source code showing the application of VR-based multimodal interaction concepts.

3.4.6 Hello, VR^{2}S

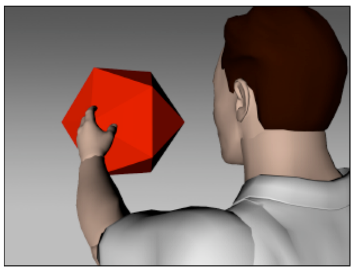
In this section, a “Hello, VR^{2}S” example illustrates the usage of multimodal interaction concepts in VR^{2}S applications. Listings 3.6 shows a minimal application involving a virtual input device, represented by a sphere, which can be controlled with an arbitrary optical tracked device. At the position of this virtual input device a sound occurs. Also head tracking is activated to enable exploration of this minimal scene. Furthermore the haptic input device is appended to the behavior graph.

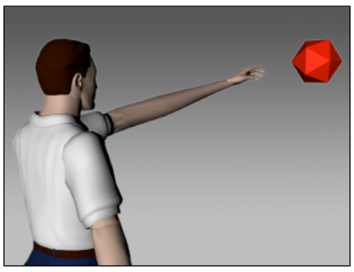
Chapter 4

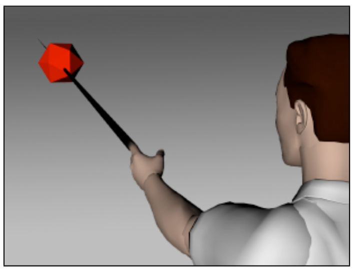
Selection and Manipulation of Virtual Objects

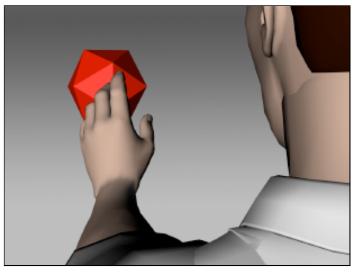
Although virtual environments have shown considerable potential for providing intuitive interfaces for human-computer interaction, improving the acceptance of VR technology requires optimization of the most basic interaction tasks to maximize user performance and provide efficient HCI. Before interacting with virtual objects, the user needs to specify the target for the desired interaction. This *selection* is generally considered as an interaction task itself. To perform an object selection a set of *selectable objects*, a technique for identifying the object to be selected and a mechanism to indicate the time of selection are required. Furthermore, as described in Chapter 2, the user should get adequate multimodal feedback about a possible or an already performed object selection, e.g., *target feedback* that indicates which object will be selected, for instance by highlighting the object or displaying the object's bounding box. When an object is selected six DoF manipulation of the selected object can be performed to change the object's position and orientation until the object is released. Besides *local selection*, i.e., selection in the immediate reach of the user, also *distant objects* can be manipulated when using VR interaction techniques. As described in Chapter 2, such VR-based interactions put a high cognitive effort on the user. Metaphors are introduced to abstract from the complexity of VR-based interactions. This chapter discusses direct interaction metaphors for object selection and manipulation in VEs.

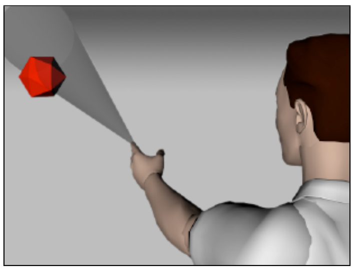
The chapter is structured as follows. Section 4.1 discusses related VR-based interaction metaphors supporting the user, when performing object selection and six DoF manipulation in a direct way. Section 4.2 describes an enhancement of these metaphors – the improved virtual pointer metaphor – in detail and explains its integration into VR^{2}S as well as the results of a usability study comparing these metaphors. A dual-purpose metaphor for both VR-based as well as desktop-based interaction is described in Section 4.3. Section 4.4 sketches the integration of VR widgets into VR^{2}S for indirect interaction.


(a) Virtual hand metaphor


(b) Go-go metaphor


(c) Virtual pointer metaphor


(d) Sticky-finger metaphor


(e) Spotlight metaphor

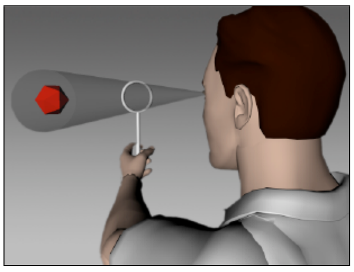

(f) Aperture-based metaphor

Figure 4.1: Different direct interaction metaphors for VR-based interaction.

4.1 Direct Interaction Metaphors

With the increasing availability of VR technologies, many basic approaches for direct interaction metaphors in VEs have been proposed. In [BH97] BOWMAN ET AL. have evaluated and compared several selection and manipulation metaphors, among them the *virtual hand* and the *virtual pointer metaphor*. Both metaphors use a virtual input device, which is controlled by a real input device. They differ in the way selections and manipulations are performed.

When using the virtual hand metaphor, a selection is possible when the virtual input device intersects the desired object (see Figure 4.1 (a)). If the desired object is selected manipulations can be performed as in the real world by a one-to-one mapping between the movements of the virtual input device and the virtual object. Thus interaction is performed as if the virtual object were located in the user's hand. Although this approach is straightforward and very natural it limits the potential of VR interactions, since selection of distant objects outside the immediate reach of the user is not supported and has to be realized by applying alternative strategies.

The *go-go metaphor* ([PBWI96]) supports selection of distant objects by using a nonlinear mapping function that transfers the physically measured distance between the user's head and his hand to the controlled distance between the real and the virtual hand. Thus, with increasing distance between the real hand and the head of the user, the movements of the virtual input device are scaled with respect to the movements of the real hand. This relation between the movements of the real and the virtual hand is called *control/distance (C/D) ratio*. When the user navigates the virtual hand to the desired object, selection and manipulations can be performed by a one-to-one mapping as in the classical virtual hand metaphor. The go-go metaphor is illustrated in

Figure 4.1 (b).

In contrast to moving a virtual hand to the object, when using virtual pointer metaphors, e.g., casting a ray through the VE ([PFC+97, JFH94, Fit54]), a selection is performed when the virtual ray hits the desired object as illustrated in Figure 4.1 (c).

Image plane interaction techniques involve pointing at the 2D image plane as done in the *sticky-finger metaphor* ([PFC+97]); here the object underneath the user's finger is the one which can be selected (see Figure 4.1 (d)). This intuitive form of selection is similar to using a mouse cursor for two-dimensional interfaces, but it leads to problems when selecting distant and small objects, since their projection is usually too small to be hit by the ray. Moreover, hand tremors and accuracy errors of tracking systems complicate the interaction with distant objects.

Poupyrev et al. compared in [PWBI98] the go-go and a ray-casting technique, which show comparable performance for local selection conditions, independent of the virtual object's size. With increasing distance, especially when higher selection accuracy is required, the go-go metaphor has a significant performance advantage.

Accuracy errors occurring when using the virtual pointer metaphor can be reduced by using a cone instead of a ray, as it is done in the *spotlight metaphor* ([LG94]) (see Figure 4.1 (e)). But several objects may fall into the cone. In [FHZ96] a modification of the spotlight technique is described, which reduces these ambiguities by providing aperture-based and resizable selection cones as depicted in Figure 4.1 (f). To select fully or partially occluded objects Olwal and Feiner ([OF03]) have described a *flexible pointer* visualized by a curve. This approach is based on a two-handed control of the curve, it uses the vector formed by the hands to determine the pointer's direction and the orientation of each hand to control the amount of curvature.

In addition to the problems of accurate selection of distant or small objects, the most often mentioned drawbacks of virtual pointer metaphors are the difficulties occurring during six DoF manipulations of virtual objects. Such manipulations may change the orientation of the ray that connects the virtual input device to the selected object. Thus the center of manipulation is not located in the center of the object anymore, hence in particular rotations around a different axis than the ray are difficult to perform.

For this reason, Bowman et al. [BH97] have proposed a hybrid approach, called *HOMER* (*hand orientated manipulation extending ray-casting*), a technique combining virtual pointer and virtual hand metaphors. For selection of both local as well as distant objects, the intuitive ray-casting technique is used. After the selection is performed, the virtual input device moves to the desired object, or vice versa. Now manipulations can be realized as when using the classical virtual hand metaphor. After finishing the desired manipulations the virtual object is moved back to its initial position. The HOMER approach has two disadvantages: when moving the virtual input device to the selected object, the object is obscured, and when moving the object to the virtual input device, the spatial relationships of the scene are changed.

4.2 Improved Virtual Pointer Metaphor

Many VR-based applications have shown that virtual pointer metaphors are intuitive for both local and distant direct object interaction. Besides their natural usage, however, virtual pointer metaphors need to be improved with respect to the way of aiming at virtual objects and performing six DoF manipulations.

In order to achieve these goals the improved virtual pointer (IVP) metaphor, which avoids most of the aforementioned disadvantages of current direct interaction metaphors, has been introduced ([SRH04]). This approach allows the user to select a desired object without requiring an exact hit with the ray. A straight ray is used to indicate the direction of the virtual pointer, while an additionally visualized bendable ray snaps to the closest selectable object (see Figure 4.2).

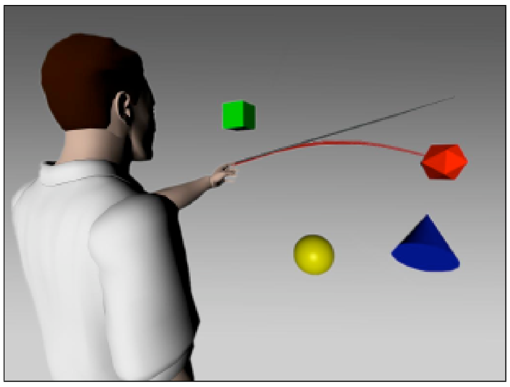


Figure 4.2: Illustration of the IVP metaphor.

The closest selectable object that will be chosen if the user performs a selection, e.g., by pressing a button or by pinching a glove, is called *active object*. After selecting the active object, successive manipulations can be accomplished by different mapping strategies between the movements of the input device and the manipulated object.

The improved virtual pointer advances direct object interaction by combining the following advantages:

- simple selection of local as well as distant objects,
- prevention of accuracy errors and ambiguities,
- sufficiency of two DoF for controlling the virtual input device in many application areas,
- possible selection of occluded objects, and
- intuitive six DoF manipulations of local as well as distant objects.

4.2.1 Selection Process

As mentioned above, ray-casting approaches require a simple way to aim at virtual objects. The IVP metaphor combines the advantages of the metaphors described in Section 4.1 and extends them to provide an intuitive mechanism for direct object interaction. The metaphor enhances the selection process of an object, i.e., with the IVP metaphor a user performs a direct object selection by roughly pointing at the desired object. Thereupon the flexible ray bends to this active object if it is the one closest to the straight ray. In the following subsections, different approaches are described how to determine the active object.

Distance Calculation

Usually the desired object a user wants to select is the one closest to the ray. For an improved object selection a straightforward approach is to choose the object with the minimal orthogonal distance to the virtual ray. This minimal orthogonal distance may refer to different reference points of a desired object, e.g., the center of the object's bounding box, the nearest vertex, the nearest edge etc. ([DFS01]). If more than one object have the same minimal distance to the ray, different strategies may be considered, e.g., the object closest to the user becomes active.

To determine the closest object for all considered objects, their distances to the virtual ray have to be calculated. The *ActiveObjectList* (AOL) is an ordered list that stores all selectable objects in increasing order of their distances to the virtual ray. The first object with minimal orthogonal distance is the active object: AOL=$\{(obj_1,d_1),...,(obj_n,d_n)\}$ with $0 \leq |d_1| \leq ... \leq |d_n|$, and obj_1 is the active object. This list provides the possibility to switch between active objects, e.g., to solve problems occurring during the selection of partially or fully occluded objects.

World Distance

The minimal orthogonal distance can be calculated by dropping a perpendicular from the reference points of all considered objects to the virtual ray.

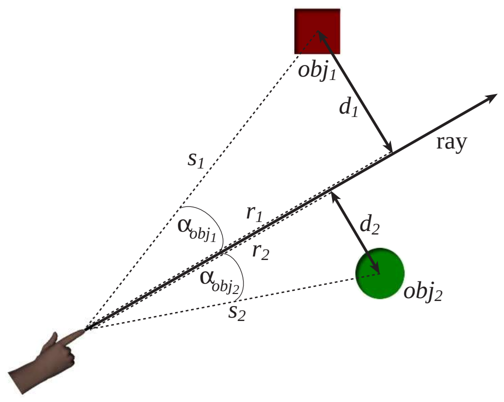


Figure 4.3: Calculation of the distances between virtual objects and the ray.

Figure 4.3 illustrates how the distance d_i from the virtual ray to a reference point of an object obj_i is calculated using the ray direction, the vector s_i from the position of the virtual input device to the reference point of obj_i, the vector r_i from the position of the virtual input device to the intersection point of the ray with the plane that is orthogonal to the ray and contains the reference point of the considered object, and the angle α_{obj_i} between s_i and r_i. The angle α_{obj_i} and the distance d_i can be calculated as follows:

$$\begin{aligned} \alpha_{obj_i} &= \arccos\left(\frac{s_i \cdot r_i}{|s_i| \cdot |r_i|}\right) && (4.1) \\ d_i &= \sin\left(\alpha_{obj_i}\right) \cdot |s_i|. && (4.2) \end{aligned}$$

Ordering the objects within the *ActiveObjectList* according to this world distance may cause results not expected by the user, since the displayed distance may be distorted because of the perspective projection transformation (see Chapter 3).

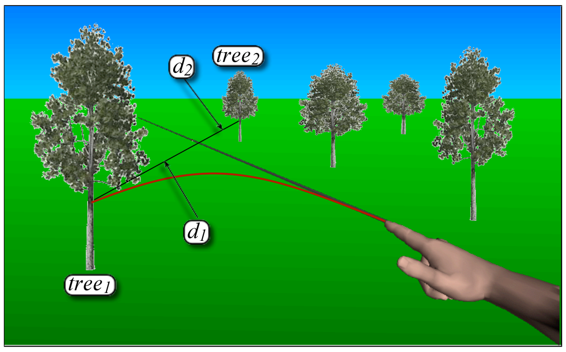


Figure 4.4: Perspective distortion when selecting distant virtual objects.

Figure 4.4 illustrates this problem. From the user's point of view, tree number 2 ($tree_2$) seems to be closer to the ray. However, tree number 1 ($tree_1$) attracts the curve and gets active since the distance between $tree_1$ and the ray is less than the distance between the ray and $tree_2$, even though d_1 seems to be larger because of the perspective distortion. The image plane distance approach and the conic extension approach which will be described in the following do not have this drawback ([SRH05c]).

Image Plane Distance

This approach calculates the distances used for ordering the objects in the *ActiveObjectList* in image space coordinates. The world space distance d_i is transformed into the corresponding image space distance id_i as illustrated in Figure 4.5.

The Figure shows that even a large world space distance may appear quite short after the transformation. Although object obj_2 is located farther from the ray than obj_1, its projected distance id_2 is shorter than id_1. Thus ordering the *ActiveObjectList* according to image space distances may also lead to unsatisfying results.

Conic Extension

This alternative approach considers both the world distance between a particular object obj_i and the ray, and the distance between the virtual input device and the object. The world distance is

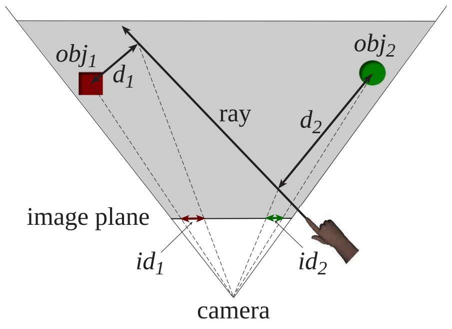


Figure 4.5: Selection on the basis of distances in image space.

multiplied by a scale factor $f_i := \frac{1}{|r_i|}$, and the resulting value is used to order the objects stored in the *ActiveObjectList*. The scale factor f_i is the inverse of the length of the vector r_i introduced before (see Figure 4.3). Hence f_i decreases with increasing distance between the virtual input device and a virtual object obj_i, and multiplying the world distance with this factor yields a smaller value as basis for ordering the *ActiveObjectList*.

Although both approaches use different values for ordering the objects both provide access to the world distances.

Sticky-Ray Metaphor

In a densely populated VE with large objects and small gaps between them distance calculations may be unnecessary, because most times the virtual ray hits a virtual object. Only in the small gaps between the objects, distance calculations would be necessary. Therefore the following strategy may be advantageous. Like for the simple virtual pointer metaphor a ray is cast through the VE and the first object to be hit becomes the active object. It remains active until the virtual ray hits a different selectable object. This way selection is simplified because only a single hit of a desired object with the selection ray is needed to ensure a selection. This strategy leads to the *sticky-ray metaphor* ([SRH04]) that is illustrated in Figure 4.6.

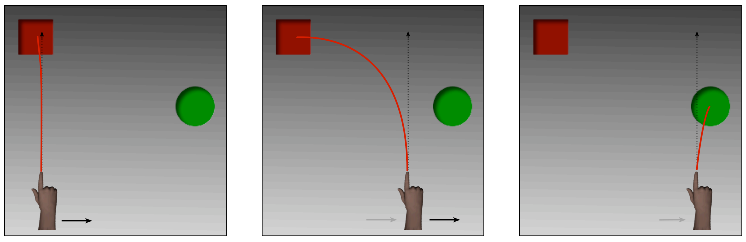


Figure 4.6: Illustration of the sticky-ray metaphor during a translation of a virtual input device from left to right.

The process depicted in Figure 4.6 shows a dynamic scene illustrating the movement of a virtual input device from the left to the right. In the beginning the ray intersects the red box leading to a feasible selection (left). When moving to the right, the red box remains active (indicated by the red curve) although the green sphere is closer to the virtual ray (middle). Moving further to the right, the ray hits the green sphere, which then becomes the active object (right). In contrast to the concepts described before the sticky-ray metaphor needs no distance calculations at all, only a ray-cast is evaluated for the selectable objects.

Region Examination

When determining the active object from a set of objects, the number of considered objects depends on the scene structure and the examined region. When the user moves the virtual pointer through the VE, within an appropriate region the object hit by or closest to the virtual ray has to be determined. The optimal choice for structure and size of this region depends on the scene's configuration, i.e., the number and arrangement of the selectable objects within the VE. Two possible strategies can be distinguished:

1. Reducing the number of considered objects, e.g., depending on intersections with geometric shapes, or

2. considering all selectable objects.

In the first case only objects intersected by a predefined geometric shape are considered. Possible geometric shapes are cones, cylinders, spheres, boxes etc., which may be attached to the virtual input device or located somewhere in the VE. Those shapes are not visualized, they are only used for reducing the number of objects to be considered in calculations. If one or more selectable objects intersect such a geometric shape, the object with the smallest distance to the ray becomes the active object. If no such intersecting object exists, the geometric shape is enlarged and tested again for intersecting objects. This enlargement process is repeated until either an intersection is found or the complete scene has been examined without success, i.e., the scene does not contain any selectable objects.

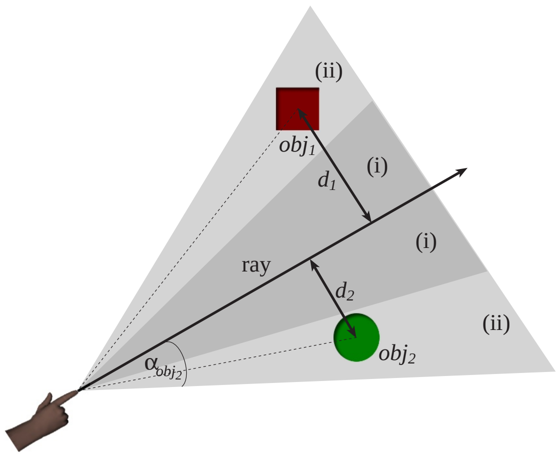


Figure 4.7: Example configuration with initial (i) and extended (ii) cone.

Figure 4.7 illustrates an example. Since the virtual ray does not hit any object, an initial cone-shaped region (i) is examined. Because this cone intersects no objects a larger cone is analyzed. Now two objects intersect the cone (ii) and due to $|d_2| < |d_1|$, obj_2 becomes the active object. This strategy is favorable for extremely densely populated scenes, but the appropriate size of the region to be examined depends on the topology of the scene.

Alternatively, the distances for all selectable objects can be calculated. This strategy may be advantageous in VEs in which the initially examined region usually does not contain any objects. Thus multiple examinations of regions can be avoided at the cost of calculating in one scene traversal for all selectable objects their distances to the ray. However, since the described distance metric is not very complex, this approach is sufficient for most VEs. These strategies do not require properties of the data structures used by the VE, for instance, an octree, but they are adaptable to such concepts.

Visualization of the Virtual Ray

An adequate visual feedback of a possible selection needs to take the ray direction vector as well as the position of the active object into account. This is ensured by visualizing in addition to the ray direction vector a *Beziér curve graph*

$$B(x) = \sum_{i=0}^{2} p_i \cdot \begin{pmatrix} 2 \\ i \end{pmatrix} x^i (1-x)^{2-i}, \tag{4.3}$$

with $x \in \mathbb{R}^3$ and three points $p_i \in \mathbb{R}^3$, $i = 0, ..., 2$, defining the curve. The anchor points p_0 and p_2 are defined by the position of the virtual input device and the active object's reference point, e.g., the center of its bounding box. The control point p_1 is located on the ray direction vector and determines the bending of the Beziér curve.

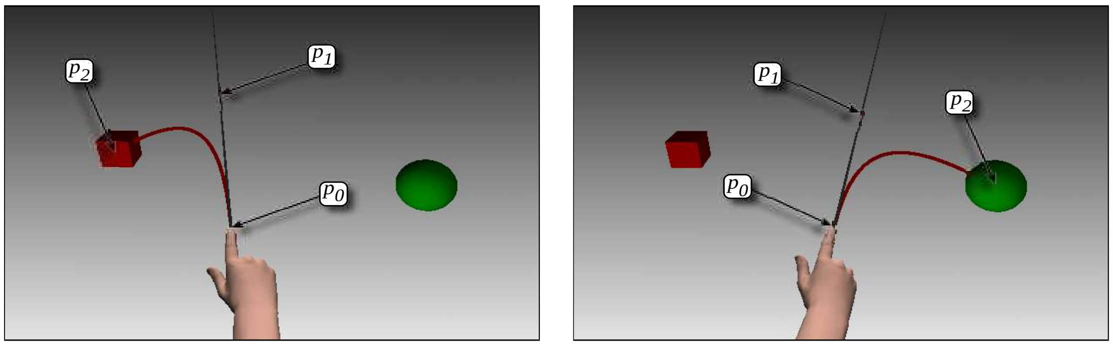


Figure 4.8: Illustration of the bending of the curve with respect to the anchor points p_0, p_2 and the control point p_1.

Tests have indicated that $0 < |p_0 - p_1| < |p_0 - p_2|$ should be satisfied, otherwise the attraction is too small or appears to be unnatural. Figure 4.8 shows a virtual scene illustrating the curve for two selectable objects with p_1 chosen such that $|p_0 - p_1| = \frac{4}{5} \cdot |p_0 - p_2|$. The red box is active although it is not hit by the virtual ray (left). After a small rotation of the virtual hand the green sphere is located closer to the ray and becomes the active object (right).

4.2.2 Manipulation of Virtual Objects

This section describes how to overcome the aforementioned drawbacks of distant object manipulation, in particular rotation around an arbitrary axis, when using virtual pointer metaphors. The IVP metaphor extends the idea of the previously described HOMER technique to provide an efficient tool for six DoF direct manipulations of both local as well as distant virtual objects. In contrast to the HOMER technique, the virtual input device and the selected object both remain at their initial position after a selection is performed. Nevertheless, all rotations are implemented by a one-to-one mapping between the rotational movements of the virtual input device and the virtual object. Thus rotations can be performed similar to real world rotations, except that the manipulated object remains at its original position without being relocated towards the user's hand. By using this approach manipulations, in particular rotations, can be accomplished accurately without occluding the desired object by the virtual input device, as it may happen when using the HOMER technique.

For translational movements of the virtual input device the one-to-one mapping is maintained, but in addition a linear mapping function between the movements of the virtual pointer and the manipulated object is applied. This is done since in some cases perspective distortion may scale the movements of the virtual input device depending on the position and distance between the input device and the selected object.

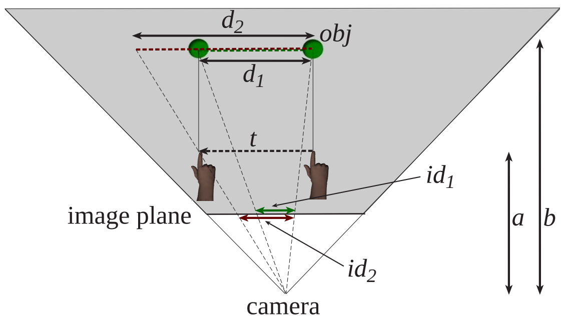


Figure 4.9: Distant object translation affected by perspective distortion.

Figure 4.9 clarifies this issue. Mapping a translation t of a virtual input device, which is located close to the viewpoint, one-to-one to a remotely selected object obj results in a translation d_1 that appears to be foreshortened (compare d_1 to d_2). For this reason, translational movements are scaled with the factor $t_\mathrm{f} = \frac{b}{a}$ where a denotes the distance between the camera and the virtual input device, and b the distance between the camera and the desired object (see Figure 4.9). Small and accurate translation of distant objects is more complicated using this approach, but existing VR applications have revealed that accurate and precise manipulations are accomplished primarily by local interaction within the immediate reach of the user. In the case of local object translation $a \approx b$ and therefore $t_\mathrm{f} \approx 1$. Hence, translations of local objects are barely affected.

Direct six DoF manipulations of distant objects are mostly used for moving these objects close to the user for exploration or for performing larger translations. When the virtual input device is farther away from the user than the manipulated object, the movements of the virtual object are downscaled in relation to the movements of the virtual input device, and precise manipulations can be performed. Hence, both aspects, precise and rapid interactions, are incorporated into this approach.

4.2.3 Multimodal Concepts

In this section the adaptation of multimodal interaction concepts to fit the needs of object selection and manipulation in VEs is described. In particular it is shown, how using multimodal input as well as multimodal output can enhance the interaction process.

Multimodal Input

Two-handed interaction can be used to improve object interaction. In realizations of pointer metaphors ideally wands or other pointing devices are used with the dominant hand, whereas data gloves or other devices can be used with the non-dominant hand for system control. For example, the haptic input device described in Chapter 3 can be used to control the virtual input device according to the IVP metaphor by pointing at the desired object as described in the previous subsections. A selection is indicated by pressing a button on this device. Other input devices, such as a glove, can be used by the non-dominant hand to access menus or to support object selection. In case the desired object is occluded, a data glove can be used for traversing the *ActiveObjectList* in order to select a different virtual object.

Multimodal Output

Multimodal feedback can be used to inform the user about a possible selection. Furthermore, multimodal feedback can encode data generated within the VE in a way that it is intuitively comprehensible for users. For this reason, visual, auditory and haptic senses are addressed. As described above, adequate visual feedback of a possible selection is ensured by visualizing, in addition to the ray direction vector, the Beziér curve pointing to the active object.

To improve the user's perception of the active object, acoustic information is submitted to the user when the active object switches. When by moving the virtual input device a different selectable object gets active, i.e., the Beziér curve bends to the new active object, the position and orientation of both the active object and the user are used as parameters for the sonification process. A switch of the active object can be emphasized by a gentle sound dispersing from the position of the active object towards the user's position. As a result, the active object can be spatially located more easily.

In addition, haptic feedback is incorporated for further improvement of direct object interaction techniques, i.e., in addition to the visual and acoustic cues, the user gets haptic information regarding the active object. During a switch of the active object, the user receives a light and short vibration signal emitted by the haptic input device. The intensity of vibration can be altered depending on the distance between the virtual input device and the active object. Starting from an initial low intensity of vibration, a decreasing distance between input device and active object results in a higher intensity of vibration. Thus the usage of haptic feedback has the potential to enhance comprehension about the arrangement of the selectable objects, in particular the active object.

4.2.4 Integration of the IVP Metaphor into VR^{2}S

In this section the integration of the IVP metaphor into VR^{2}S is described. Since the IVP metaphor is a special metaphor to control virtual input devices, it is implemented in the class `ImprovedVirtualPointer`, derived from the class `SpatialInputDevice` explained in Chapter 3 (see Figure 4.10). An instance of the type `ImprovedVirtualPointer` is associated to a scene node, which corresponds to the visual representation of a virtual input device such as a virtual hand. The IVP interaction concepts described in the previous subsections to select, translate, rotate and release virtual objects are implemented in four corresponding callback methods, i.e., `motionIVP`, `selectIVP`, `manipulateIVP` and `releaseIVP`. The argument passed to these callbacks is an object of type `IntersectionInfo` that is obtained by executing a ray query, which returns the shapes hit by the virtual pointer resp. a null object if no shape is hit.

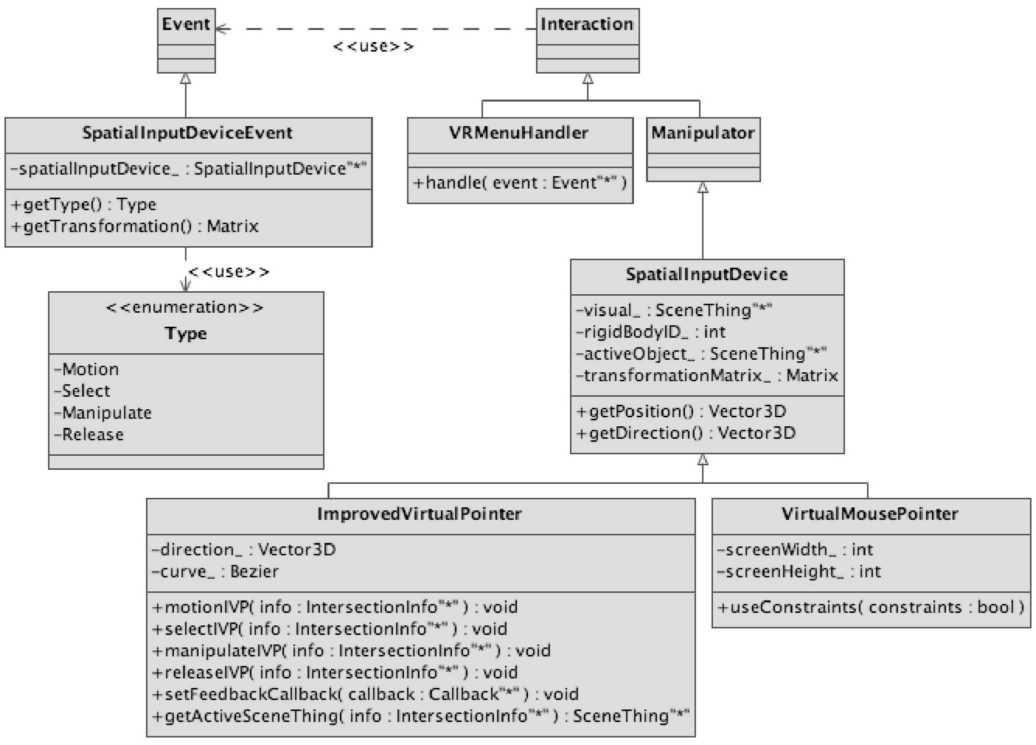

Event
<<use>>
Interaction
SpatialInputDeviceEvent
-spatialInputDevice_ : SpatialInputDevice"*"
+getType() : Type
+getTransformation() : Matrix
VRMenuHandler
+handle(event : Event"*")
Manipulator
<<use>>
<<enumeration>>
Type
-Motion
-Select
-Manipulate
-Release
SpatialInputDevice
-visual_ : SceneThing"*"
-rigidBodyID_ : int
-activeObject_ : SceneThing"*"
-transformationMatrix_ : Matrix
+getPosition() : Vector3D
+getDirection() : Vector3D
ImprovedVirtualPointer
-direction_ : Vector3D
-curve_ : Bezier
+motionIVP(info : IntersectionInfo"*") : void
+selectIVP(info : IntersectionInfo"*") : void
+manipulateIVP(info : IntersectionInfo"*") : void
+releaseIVP(info : IntersectionInfo"*") : void
+setFeedbackCallback(callback : Callback"*") : void
+getActiveSceneThing(info : IntersectionInfo"*") : SceneThing"*"
VirtualMousePointer
-screenWidth_ : int
-screenHeight_ : int
+useConstraints(constraints : bool)

Figure 4.10: Classes used to integrate the IVP metaphor into VR^{2}S.

Besides the different mapping and distance calculation strategies proposed in Sections 4.2.1

and 4.2.2, the visual representation of the curve can be altered in terms of bending, color and size by corresponding attributes of the method `setCurveStyle`. The method `setFeedbackCallback` enables the application developer to specify multimodal feedback, which is submitted to the user when the active object changes. The application developer defines which virtual objects can be selected and arranges them in the VE by using corresponding scene nodes. The scene node associated with the active object can be inquired with the method `getActiveSceneThing`. Determination of the active object is a major task of the IVP metaphor and is achieved by computing the distances of all selectable objects to the virtual ray as described in Section 4.2.1. The scenegraph structure used in high-level graphic systems can be exploited to improve performance by calculating all distances during a pre-evaluation phase of the scenegraph when using the previously described multipass rendering concepts (see Chapter 3). For this purpose, the pass `ACTIVEOBJECTTECHNIQUE` is used (see Table 3.1). After the bounding boxes of all selectable objects have been determined in the pass `BOUNDINGBOX`, within this pass the distances between these objects and the virtual ray are calculated and the *ActiveObjectList* is generated i.e., these objects are stored in the *ActiveObjectList* according to their distances to the ray. When an instance of type `ActiveObjectTechnique` is appended to the scenegraph, the calculation of the distances as well as creation of the *ActiveObjectList* is initiated. The `prepareEval` method obtains as parameters a shape and a VRS engine for evaluating the shape. `prepareEval` checks the shape wether it contains a flag which indicates that this shape resp. the associated scene node is selectable.

The application developer has to specify the shapes or scene nodes that can be selected. For these selectable objects the distance calculations can be performed using Equations (4.1) and (4.2).

The results of these simple distance calculations may be scaled according to the approaches described in Section 4.2.1. The final results are stored in the ordered *ActiveObjectList*. This list can be used, for example, to switch between selectable objects, e.g., to select an occluded object by traversing the ordered list. If the active object changes, the callback mechanism initiates auditory and haptic feedback as described in Chapter 3 by associating a sound source to the active object and sending a vibration signal which depends on the distance between the user and the reference point of the active object.

In Listing 4.1 the application of the IVP metaphor within a VR^{2}S example application is shown. In line 2 an instance of the class `ActiveObjectTechnique` is appended to the scenegraph and an improved virtual pointer is set up in lines 4-8, i.e., the virtual hand model depicted in Figure 4.8 is inserted into the scenegraph, and it is declared as an improved virtual pointer. Hence a Beziér curve is appended to the visual representation of the virtual hand. The method callback `activeSceneThingCb` returns the active scene node.

Frame Rate Comparison

When embedding all previously described distance calculations into a pre-evaluation traversal of the scenegraph, interactive frame rates are maintained for reasonably populated virtual environments.

```
...
main->append(new ActiveObjectTechnique());

SO<SceneThing> virtualHand = new SceneThing();
virtualHand->append(ThreeDSReader::readScene("hand.3ds"));

SO<ImprovedVirtualPointer> ivp;
ivp = new ImprovedVirtualPointer(camera, virtualHand, activeSceneThingCb);

canvas->append(ivp);
...
```

Listing 4.1: Using the IVP metaphor in VR^{2}S.

	frames per second	
|selectable objects|	**Ray-casting**	**IVP**
50	≈ 47	≈ 46
100	≈ 26	≈ 25
500	≈ 6	≈ 5

Table 4.1: Frame rates for the IVP metaphor in comparison to a simple ray-casting approach.

Table 4.1 shows that even for an increasing number of selectable objects, almost the same frame rates can be maintained for the IVP metaphor as achieved when using a simple ray-casting metaphor without any distance calculation. The example scene chosen for this frame rate comparison consisted of different textured and colored randomly arranged spheres. For sufficiently large frame rates the difference does not matter, but for small frame rates users may perceive the slowdown. The frame rates have been measured on an Apple G4 dual processor system equipped with two 1.25GHz processors, 1GB RAM and an *n*VIDIA GeForce 4Ti graphics card.

4.2.5 Usability Study

To evaluate the concepts of the IVP metaphor, a usability study has been performed. During these tests the subjects had to accomplish several selection and positioning tasks using the different interaction metaphors explained in Section 4.1. The time required as well as the accuracy for each subtask have been measured, and a user survey about the interaction concepts has been analyzed.

Tasks

As basis for the evaluation of the IVP metaphor a residential city planning environment has been chosen as illustrated in Figure 4.11. The tasks consisted of a combination of selections and

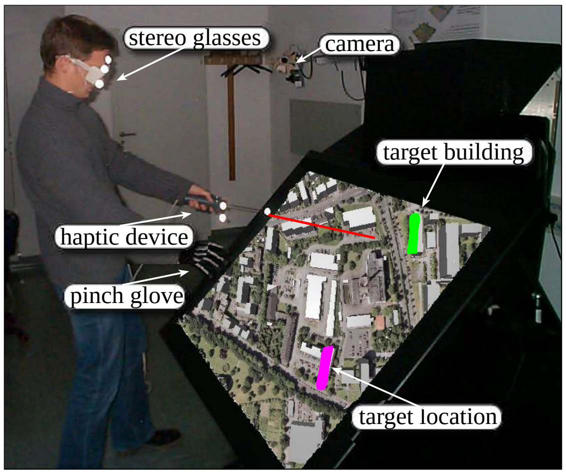


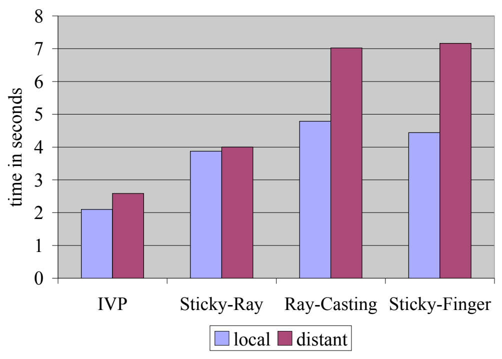
Figure 4.11: Usability test environment: responsive workbench, optical tracking system, haptic input device and pinch glove.

manipulations. Several randomly highlighted buildings had to be selected and repositioned to a particular marked location. After a short introduction into the usage of the metaphors, during the test phase the times needed for both selection and manipulation as well as the accuracy of the positioning task were measured. The IVP and sticky-ray metaphor were compared to the most common interaction metaphors supporting both local as well as distant object interaction, i.e., simple ray-casting metaphor and sticky-finger metaphor (see Section 4.1). The effect of multimodal interaction was measured in terms of performance and it has been considered afterwards in a survey. The 11 male and 4 female subjects chosen for these tests were familiar with residential city planning environments. Most subjects were geoinformatic students (8), but also computer scientists (3), mathematicians (2) and landscape ecologists (2) participated in the tests.

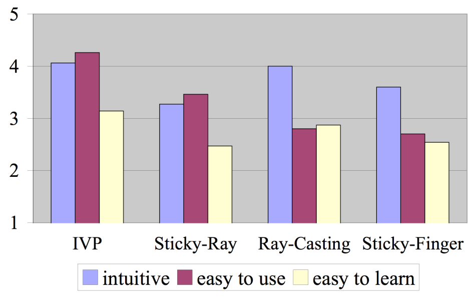
Results

The results of the user study show that the IVP metaphor has significant advantages in comparison to the other metaphors. Most striking is the improvement for the selection subtask when using the IVP or sticky-ray metaphor. Figure 4.12 (a) shows the times needed for the selection subtasks. Local and especially distant selection can be accomplished faster by using the IVP metaphor as well as the sticky-ray metaphor. A statistical analysis has revealed that the hypothesis which states that selection using the IVP metaphor reduces the required time can be assumed as true with a level of significance of 95% ([Ebe94]).

A comparison between the world space distance calculation and the conic extension approach (see Section 4.2.1) shows that the conic extension approach leads to a significant performance gain when selecting objects. Using world space distance calculation an object selection took on average 2.49 seconds, using the conic extension approach it took 2.00 seconds on average. The proposed scaled distance manipulations have already been evaluated in [FK05] and showed



(a) Needed time for local and distant selection tasks.



(b) Results of the user survey

Figure 4.12: Results of the usability study.

better performance in comparison to one-to-one mapping approaches.

Figure 4.12 (b) presents a part of the results of the user survey. The survey shows that the users preferred interaction using the IVP metaphor. The users had to assess how intuitive and easy to use the metaphors are, and they had to judge the learning effort. The evaluation has been performed on a *five-point Likert scale* ([Lik32]), where grade 1 corresponds to not intuitive, difficult to use and hard to learn, while grade 5 corresponds to the opposite. Although the differences are not as significant as the results for the selection task, the users rate the IVP metaphor as the most intuitive and easy to use metaphor. In the users' view, the learning effort for the IVP is minimal in comparison to the other metaphors. Furthermore, the survey has shown that the participants consider the use of multimodal feedback as very helpful, especially the haptic feedback emitted when a new object becomes active. Although the usage of both auditory as well as haptic feedback has not increased the performance in this usability study, the survey and comments of the participants have shown that multimodal feedback improves the selection process in so far as the users feel safe and confirmed when they select an object. The participants rate acoustic feedback on average with 3.5 on a five-point Likert scale, where grade 1 corresponds to not helpful, while grade 5 corresponds to very helpful. Haptic feedback has been evaluated with 4.2 on average; especially female participants prefer the support by haptic feedback (4.0 on average) in comparison to acoustic feedback (3.0 on average). Moreover, the haptic input device described in Chapter 3 has been assessed as ergonomically designed in terms of weight and handling comfort.

Currently, the concepts underlying the IVP metaphor are also used at the Scientific Visualization research group at the Delft University of Technology ([dHKP05]).

4.3 Dual-Purpose Interaction Metaphor

As explained in the previous sections several approaches for six DoF manipulations to interact with the VE are supported by today's VR systems. Problems arise when the user wants to perform actions not supported by the VR-based application. For example, to switch between different applications requires the user to put away the currently used interaction devices and to perform the task by using a desktop mouse. For the feeling of immersion it is very important that the user does not have to leave the VR system for operations like reconfiguration of the system, starting a new session etc., and then enter it again.

In this section an approach is described which enables users of VR systems to switch seamlessly between desktop-based and VR-based interactions. This *dual-purpose interaction metaphor* ([SRH05d]) is based on laser pen interaction ([ON01]) and enables the user to perform six DoF interactions as well as controlling the 2D desktop within the VR system. In particular, relatively small desktop widgets can be accessed using a non-2D input device. Thus it is possible to control applications with graphical user interfaces that were not developed with the intention to be used in an immersive VR environment.

4.3.1 Desktop-based Interaction

The adaptation of 3D desktop-based applications to VR systems often fails since no techniques are available that allow the user to access the full functionality of the application; especially menu interaction and system control is not implemented or must be completely redesigned for the usage in VR environments. Furthermore, when working with a VR system direct access to the operating system or another desktop-based application may be required, e.g., to use WIMP-based systems. Since standard input devices such as the mouse or the keyboard are popular and their use is familiar to most users, desktop-based interactions should rely on these intuitive mechanisms. Using the dual-purpose metaphor six DoF motions and additional signals transmitted by a VR input device are mapped to the corresponding system commands to emulate mouse input events. To calculate the current mouse cursor position a virtual ray is cast from the position of the input device towards the VR projection screen, its direction depends on the input device's orientation. The intersection point between ray and screen is transfered to the system as location for the corresponding mouse event. Thus users can handle any six DoF tracked input device in almost the same manner as a laser pen for desktop-based interaction ([ON01]).

Emulation of Mouse Motions

A user is placed in a VR system environment, e.g., in front of a projection wall, and interacts with the image plane via a laser-based input device, e.g., the haptic input device, as shown in Figure 4.13. A change of the coordinate system is necessary since the position and orientation of the input device is given in tracking system coordinates. Let T denote the matrix defining the transformation of the reference definition of the input device to its current pose calculated by the tracking system as explained in Chapter 3. From the matrix T the position *pos* of the input device in tracking system coordinates is derived by the three upper elements of the last column:

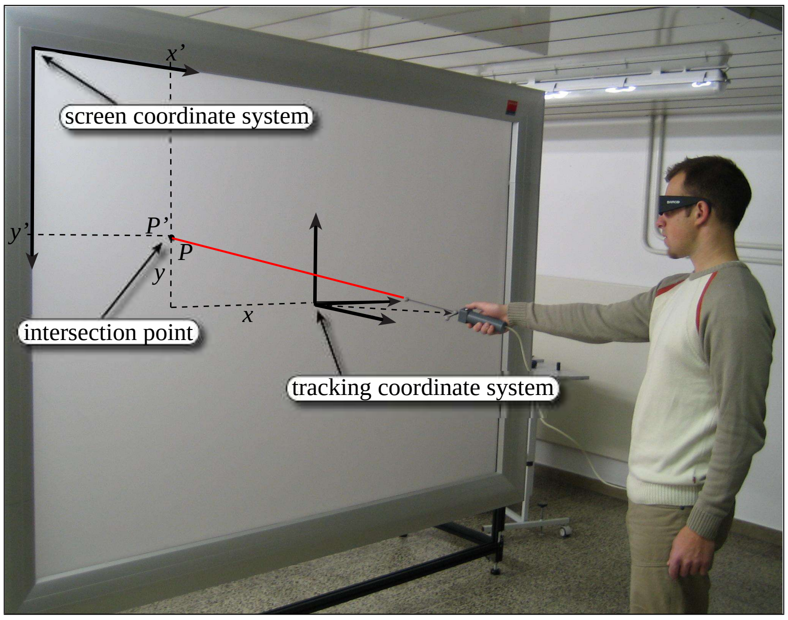


Figure 4.13: Desktop-based interaction in front of a powerwall system.

$$pos = \begin{pmatrix} T_{14} \\ T_{24} \\ T_{34} \end{pmatrix}. \tag{4.4}$$

If dir_{init} denotes the initial direction of the input device, e.g., $dir_{init} = (0, 0, -1, 1)^T$ in homogenous coordinates, which is usually derived from the reference definition of the trackable target associated with the input device, then the current direction dir_{curr} of the input device in tracking system coordinates is given by

$$dir_{curr} = \begin{pmatrix} dir_{curr_x} \\ dir_{curr_y} \\ dir_{curr_z} \\ 1 \end{pmatrix} = T \cdot \begin{pmatrix} 1 & 0 & 0 & -T_{14} \\ 0 & 1 & 0 & -T_{24} \\ 0 & 0 & 1 & -T_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot dir_{init}. \tag{4.5}$$

The tracking coordinate system and the screen coordinate system usually have the same orientation and thus the intersection point P of the ray extending from the input device in direction dir_{curr} with the x-y plane of the tracking coordinate system (see Figure 4.13) is derived by exploiting the intercept theorems as

$$P = \begin{pmatrix} pos_x \\ pos_y \\ pos_z \end{pmatrix} + \begin{pmatrix} pos_z \cdot \frac{dir_{curr_x}}{|dir_{curr_z}|} \\ pos_z \cdot \frac{dir_{curr_y}}{|dir_{curr_z}|} \\ pos_z \cdot \frac{dir_{curr_z}}{|dir_{curr_z}|} \end{pmatrix}. \tag{4.6}$$

In the case that the input device points to the projection screen, the last component of P becomes approximately zero because P is closely located to the x-y plane of the tracking as well as the screen coordinate system. Since the screen coordinate system is two-dimensional the third component of P is ignored. A transformation to the screen coordinate system, which has the dimensions of 1 to win_{size} in horizontal and vertical direction, with respect to the physical screen size $screen_{width}mm \times screen_{height}mm$ yields the intersection point $P' = (x', y')^T$ in screen coordinates:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \left\lceil \frac{win_{size}}{2} + \left(\frac{win_{size}}{2} \cdot \left(\frac{x}{screen_{width}} \right) \right) \right\rceil \\ \left\lceil \frac{win_{size}}{2} - \left(\frac{win_{size}}{2} \cdot \left(\frac{y}{screen_{height}} \right) \right) \right\rceil \end{pmatrix} \tag{4.7}$$

This point is used to generate an operating system specific mouse event at that screen location, and the mouse cursor is positioned accordingly.

Mouse Cursor Snapping

The usage of the described laser-based interaction with VR input devices involves various problems, such as hand tremor and difficulties when selecting small GUI-elements, e.g., icons and widgets ([Wis01]), which usually are designed for mouse-based interaction. To solve these problems the concepts of the IVP metaphor can be exploited and adapted to the described desktop-based interaction. Thus the IVP metaphor supports the user during the selection of small GUI-elements, which are difficult to hit with the usual pointer-based metaphor. Instead of positioning the mouse cursor at the intersection point, the cursor is placed at the position of the active widget, i.e., the GUI-element closest to the intersection point in terms of the *Euclidean distance*, and sticks there until another element gets closer to the intersection point. This snapping strategy is optional and can be activated or deactivated by the user at runtime.

Emulation of Mouse Buttons

Mouse button events can be generated by using VR input devices, e.g., left and right mouse button or double-click events are simulated by corresponding buttons of the input device or by pinching special finger combinations with gloves. Thus, the user can control all GUI-elements that can be accessed with a desktop mouse by utilizing the six DoF motions of arbitrary VR input devices, and therefore any desktop-based application can be controlled with VR devices. Furthermore special VR hardware, e.g., projection walls and space mice, can be used to perform common desktop-based interaction tasks.

Switching Modes

To allow desktop-based and VR interaction the user can switch between the VR-based interaction which uses the IVP metaphor to accomplish six DoF selection and manipulation of virtual objects and the laser-based mode for desktop-based interaction. To switch between both modes the user has to satisfy predefined conditions, e.g., pressing a special button. In addition, the modes can be switched automatically depending on the system state. For example, the activation of a two-dimensional menu can switch to the desktop-based interaction mode, while a subsequent selection of a menu entry can switch back to the VR interaction mode.

Figure 4.14 illustrates the dual-purpose metaphor in a responsive workbench environment. The user can select menu entries via the desktop-based interaction mode, and he can interact with the VE by using VR-based interaction paradigms.

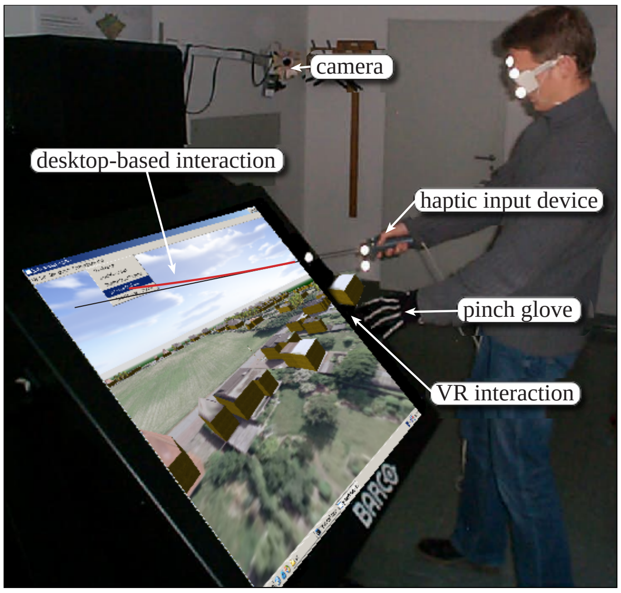


Figure 4.14: Dual-purpose interaction metaphor in front of a responsive workbench.

4.3.2 Integration of the Dual-Purpose Interaction Metaphor into VR^{2}S

The class `VirtualMousePointer` is derived from the class `SpatialInputDevice` (see Figure 4.10). It manages the steps described in Section 4.3.1 by implementing the corresponding events for mouse motions and mouse buttons in the callback methods of `SpatialInputDevice` (see Chapter 3). The six DoF data for the position and orientation of the input device are mapped to the screen position at which the mouse cursor will appear. Thus, the mouse cursor can be moved along the screen surface using any VR input device. To generate a mouse button press or release event corresponding callback methods are implemented in the class `VirtualMousePointer`.

The tracked data of a VR input device, e.g., a space mouse or the previously described haptic input device, is received via device-dependent threads that process this data and generate corresponding input events as described in Section 4.3.1. The emulated mouse events are generated by operating system specific commands, which are currently implemented for Windows platforms only and are based on the `SendInput` function of the *Windows User Interface API* ([Mic05]).

To exploit the advantages of the IVP metaphor for mouse cursor snapping, the position and region of every accessible GUI-element is needed to determine the *active element*, i.e., the closest GUI-element to the screen intersection point of the ray. Therefore, for each selectable GUI-element the distance to the intersection point has to be determined. In the image plane the Euclidian distance between the intersection point and the center of each GUI-element is used as distance metric. The position and region of each active widget is inquired from the operating system using appropriate APIs and libraries. For icons of the Windows operating system this can be achieved by using the `SendMessage` and `Readfrom` functions in conjunction with appropriate parameters (`LVM_GETITEM`, `LVM_GETITEMPOSITION`, `LVM_GETITEMRECT` etc.) ([Mic05]).

Using these concepts the dual-purpose interaction can be integrated into arbitrary VR-based applications, and thus support of both VR-based as well as desktop-based interaction within the application is guaranteed. Alternatively, the laser-based mode of the metaphor is usable without any reference to a VR-based application and can be utilized to control graphical operating systems and WIMP-based desktop applications.

4.3.3 Evaluation

The IVP metaphor concepts of the dual-purpose interaction metaphor have been evaluated and the results have been discussed in Section 4.2.5. The second part of the dual-purpose metaphor, i.e., desktop-based interaction, has been analyzed in [PNMI05]. In this evaluation study for target acquisition with pointing devices on tabletop displays a similar laser-based interaction approach has been tested. This so-called *snap-to-target selection* has been compared with several other laser-based strategies. These selection strategies are mainly based on enlarging the cursor or enlarging the target to aid the selection of small objects displayed on a tabletop system. The results of this test study underline that the desktop-based interaction concept described in Section 4.3.1 is the most efficient approach. Overall, the snap-to-target selection was the only selection aid to be perceived as significantly more comfortable and accurate than the other approaches. In addition, it also required less movement time until the target had been acquired.

4.4 Virtual Reality Widgets

Although the described direct interaction metaphors provide advanced interaction within VR systems, these metaphors lack immersive *VR system control.* VR system control is defined as the task of changing the interaction mode or the state of the VR system by issuing commands ([BKLP01]). The desktop-based interaction performed with the dual-purpose interaction metaphor allows to control desktop-based menus, but the menus are displayed monoscopically, and they may overlay stereoscopic scene content and hence disturb immersion. For this reason *VR widgets* can be used. VR widgets, sometimes referred to as *Virgets* ([MP93]), are three-dimensional analogons to two-dimensional menus. Instead of using two-dimensional menu representations, different three-dimensional shapes, e.g., boxes, spheres etc., are presented to the user associated with menus consisting, for instance, of *VR buttons*, *VR checkboxes*, or *VR sliders* ([MP93]).

4.4.1 VR Menus

VR^{2}S provides two classes, i.e., `VRWidget` derived from `SceneThing` and `VRMenuHandler` derived from `Interaction`, to realize such three-dimensional VR menus (see Figure 4.10). Objects of type `VRWidget` represent virtual widgets which can be used in VR environments. These objects contain three interchangeable scene nodes, which visually represent their possible states. Moreover, this class has a pointer to a callback function that is processed when the widget is activated. In general, two input events generated by arbitrary devices are processed, i.e., press and release events. By using these user-defined events three different states, i.e., *normal state*, *pressed state* and *checked state*, can be accessed.

1. **Normal state** is the initial state of a VR widget before a user selects this widget. In this case the scene node associated with the normal state is used to display the widget.
2. **Pressed state** represents the state to which a VR widget changes when a user clicks on the VR widget without generating a release event, e.g., releasing a button. In this state the actions associated with the VR widget are not initiated, but the visual representation of the normal state of the widget is exchanged with the representation associated with the pressed state. Thereby the user gets a visual feedback according to the chosen widget.
3. If the widget is hit by a selection ray or intersected by a virtual input device, and the user generates a release event after a select event, two alternative approaches are possible depending on the intention of the VR widget. If the scene node representing the normal state equals the scene node that represents the **checked state**, the associated callback function is applied and the VR widget returns to the normal state. This approach can be used, for instance, to provide VR buttons. In the case that the normal state and the checked state differ, the status of the VR widget changes to the checked state when the user has generated a release event. This state can be exploited, for example, to support VR checkboxes. However, if the user generates a release event while the widget is hit by a selection ray or intersected by a virtual input device, the actions associated with the desired VR widget are applied.

The management and handling of objects of the type `VRWidgets` as part of the scenegraph is ensured by an instance of the class `VRMenuHandler` derived from `Interaction` (see Chapter 3). Objects of this class ensure state changes with respect to the user's action and initiation of the associated callback function.

In Listing 4.2 the usage of the VR widgets within VR^{2}S is shown. In lines 2 to 7 the visual representations of a button and its states are defined by the application developer. In lines 9 and 10, these states are combined to an object of type `VRWidget`, which is handled by an instance of the class `VRMenuHandler` as part of the behavior graph (see line 11).

Figure 4.15 (a) shows an example of a VR menu generated within a prototype VR^{2}S application in front of a projection wall ([RSH05c]). The IVP metaphor enables an intuitive interaction with VR widgets combined to a two-dimensional menu. The menu contains several buttons, checkboxes and a slider to change the visualization properties of a volumetric PET dataset.

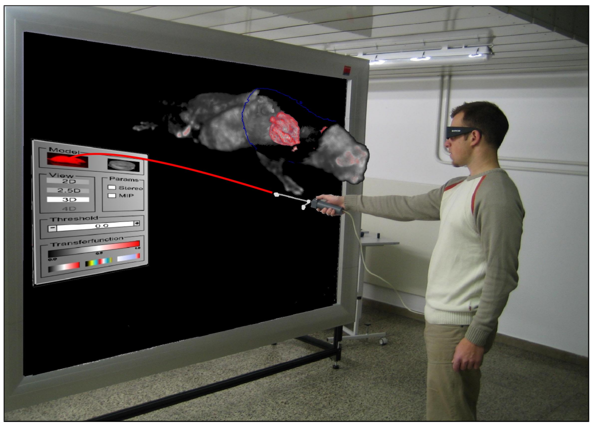
4.4.2 Personal-Interaction-Panel

A VR widget menu can also be projected onto a *personal interaction panel (PIP)*, that can be held by the non-dominant hand to enable tangible feedback during menu interaction ([SG97]).

```
...
SO<SceneThing> menu;
SO<SceneThing> widgetNormal, widgetPressed, widgetChecked;
...
widgetNormal->append(...);
widgetPressed->append(...);
widgetChecked->append(...);

menu->append(new VRWidget(widgetNormal, widgetPressed, widgetChecked,
               new FunctionCallback(&exit)));
canvas_->append(new VRMenuHandler());
...
```

Listing 4.2: Using VRWidgets in VR^{2}S.

Hence, the user can grab the PIP and touch it with another input device, such as a pen or wand. The PIP offers the possibility to manipulate all necessary controls in a desktop-based manner or as a tool-palette that groups functions and makes them easily accessible. It supports mixing the 2D desktop metaphor and a stereo display in such a way that 2D interaction and three-dimensional direct manipulation are done in parallel. Unlike many other interfaces it implements a two-dimensional interface in VR and enables 2D interaction on its surface ([SG97]).


(a) Interaction with a VR menu in front of a powerwall

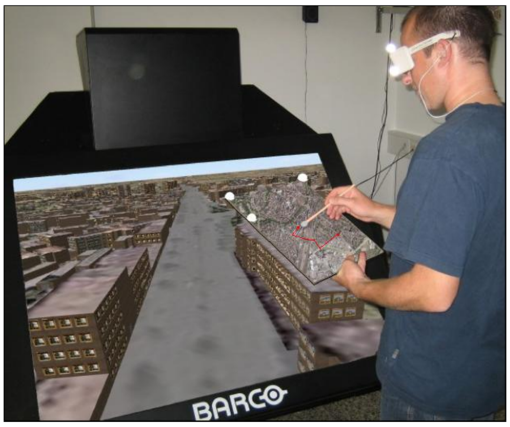

(b) Interaction with a PIP in front of a RWB

Figure 4.15: Menu-based interaction with VR widgets in VR-based applications.

The PIP is a transparent panel, which is tracked by an arbitrary tracking system. Since the system is aware of the position and orientation of the PIP, this information can be used to project a VR menu onto the projection screen in a stereoscopic way such that it appears as being projected on the PIP. The interaction with the content on the PIP can be accomplished in the same way as if the widgets are projected on the screen surface, e.g., widgets can be selected by pointing at them with a virtual pointer metaphor or pushing them with a virtual hand metaphor.

Figure 4.15 (b) shows an example of a VR menu on a PIP in a responsive workbench environment to support the user when navigating through a virtual city model ([RSH05a]).

Chapter 5

Exploration of Virtual Environments

High quality stereoscopic display technologies in combination with appropriate tracking methods allow more immersive insights and enable a better spatial comprehension when exploring virtual environments. In contrast to navigation or path-finding approaches, *exploration* is not aimed to find certain locations or paths in VEs. In fact, it denotes the more general process of investigating data with different objectives corresponding to the underlying domain. However, during an exploration process problems arise when several users want to view data in a view-dependent way, because it has to be specified whose head is tracked in order to control the virtual camera as described in Chapter 3. Furthermore, in contrast to fully-immersive VR system setups, in semi-immersive VR setups stereoscopic, view-dependent visualized data appears to be part of the real world. However, virtual objects are often independent of the properties given by the real-world surrounding, e.g., light sources or objects located in the VR work space. For instance, in current VR systems usually the light conditions have no influence on the visual representation of the virtual data, which leads to artificial representations.

In this chapter two approaches are presented which further enhance exploration in VR. Section 5.1 focusses on co-located interaction concepts for large screen displays with the objective to enable a group of users to explore virtual data with view-dependent stereoscopic visualization strategies. In Section 5.2 concepts supporting the simulation of global illumination phenomena in VR to enhance seamless merging of real-world and virtual objects are discussed. Both approaches contribute to advanced exploration of virtual datasets in VR environments.

5.1 Co-located Exploration in Projection-based Virtual Reality Systems

In real as well as virtual worlds many tasks require people to work together by sharing their knowledge and abilities to solve certain problems. Thus techniques are needed to improve effi-

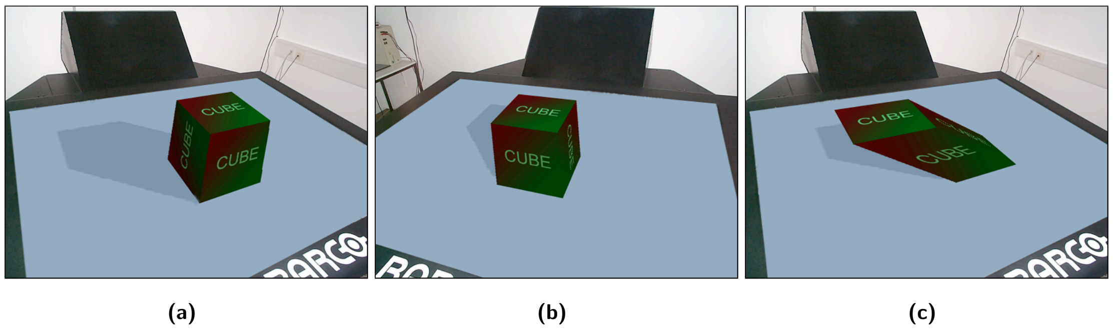


Figure 5.1: Parallax problem: (a) a cube rendered from the viewpoint of a user standing on the left side of a workbench; (b) same cube as seen by a user standing on the right side; (c) cube model seen by a user from the left side of a workbench, but rendered with the view frustum of a user standing at the right side (adapted from [ABF$^+$97]).

ciency of such group activities. In contrast to individuals working alone teams communicate and exchange information and may need to share data. For example, city planners use approximated 3D models of real-world buildings, so-called brick models, to accomplish planning tasks. In such a setup city planners can communicate and modify the position of a brick representing a building in a cooperative way. Such cooperations have been proven to be advantageous since in many application areas the bundling of experts' knowledge has the potential to increase productivity. Consequently it is desirable to develop virtual environments simulating such shared spaces in which teamwork can be performed as easily and naturally as in the real world. The vision for such *cooperative* or *collaborative virtual environments (CVE)* is to provide distributed or locally working teams with a virtual space where they can coexist, communicate and collaborate while sharing and manipulating virtual data in an interactive way.

Nowadays implementations of co-located CVEs are based mostly on individual display systems, such as see-through or fully immersive HMDs ([Sch01, BMS00]). Large projection-based display systems, for example, CAVEs, which provide fully immersive VR, or projection walls and workbenches, which provide semi-immersive VR, have considerable potential to enhance collaborative interaction. These VR system environments provide sufficient projection space to enable groups of users to see and communicate with each other face-to-face. Furthermore the excellent visual quality and wide stereoscopic field of view support working with different datasets as well as on different tasks simultaneously in a shared space. Unfortunately, collaborative interaction within these projection-based display systems is difficult, because usually these displays are capable of projecting a single stereoscopic image only. When enabling head-tracking and view-dependent rendering all cooperators but one perceive a perspectively distorted virtual scene, leading to complications in the interaction process, e.g., to control a virtual input device. Hence with respect to the interaction these projection-based environments are only single-user systems ([SS05]). In these systems rendered images are continuously updated according to the current view position and view direction of a single head-tracked user. Though non-head-tracked users are able to observe a stereoscopic image by sharing the head-tracked user's view frustum for them a comfortable interaction and exploration of the virtual scene is not possible. Since their viewpoints may differ a lot from the viewpoint of the head-tracked user. This single common viewpoint results in unexpected image motions and introduces a mismatch between the real and the virtual environment perceived by the non-head-tracked viewers ([SS05]).

Figure 5.1 illustrates the parallax problem. A cube model is displayed on a responsive workbench. Figure 5.1 (a) resp. (b) shows the cube rendered from the viewpoint of a user standing on the left resp. right side of the workbench. Figure 5.1 (c) shows that a user standing on the left side of the workbench perceives the cube model rendered with the view frustum of the user on the right side perspectively distorted. Shearing of the cube, which constricts an intuitive exploration, is clearly visible.

These issues complicate the usage of direct interaction techniques for any other than the single head-tracked user and prevent teamwork within projection-based VR systems. However, also non-head-tracked users can explore the virtual scene as long as they are placed near to the single head-tracked user, who is the only one able to interact with the VE. In the case that another user wants to participate in the interaction process, the tracked glasses and input devices must be handed over to this user, or the tracking system has to be reconfigured such that the corresponding stereo glasses are tracked to enable the user to perceive perspective-correct stereoscopic images. Now the user previously wearing the tracked glasses perceives perspectively distorted images now.

5.1.1 Related Collaborative Interaction Concepts

Since teamwork has proved itself to be efficient for many tasks in the real world, the transfer of these concepts to virtual worlds is an ongoing research issue. The objective is to enable collaborative interaction in VR system environments.

In [BMS00] Boll distinguishes between *cooperation* and *collaboration*. Collaboration describes the work performed by two or more users in parallel at the same time, while cooperation denotes the work performed by more than one user consecutively, but not in parallel. An example for collaborative interaction is the jointly lifting of a desk by more than one person, while with cooperative interaction the desk would be moved by different persons successively. Since the realization of cooperation and collaboration in VR environments is an important and challenging task, many different approaches have been proposed which try to extend existing VR systems to CVEs to provide efficient interfaces for such teamwork in both local ([Kau03, FP01, BMS00]) and distributed VR system environments ([MVS$^+$02, GLM97]). In local CVEs usually HMDs are used and thus each collaborator has his own display system ([Sch01]). Using see-through display technologies collaborators can see and communicate with each other. In distributed CVE communication and data exchange is ensured via network transfer. Avatars or video streams as representatives of remote collaborators ensure a quasi face-to-face interaction between the participants. The term *tele-immersion* denotes this kind of remote virtual face-to-face interaction over high-speed networks ([MVS$^+$02, LJB$^+$99]). Since in distributed projection-based VR systems every participant of an interaction process requires his own display system no distorted stereoscopic images occur. However, when additional users want to participate in the teamwork process either new distributed VR systems have to be set up, or distorted stereoscopic images emerge when users share projection screens with other users.

Multi-user autostereoscopic display systems enable a certain number of users to view stereoscopic images simultaneously ([Lip91]). However, the resolution and size of the displays is insufficient. Furthermore these system usually do not support head tracking.

In [ABF$^+$97] a hardware setup for a two-user collaboration in a responsive workbench environment with active stereo is introduced. Using this setup a projector displays one image for each eye of both users; the images are rendered in *user-interleaved* or *eye-interleaved* mode. In the user-interleaved mode both images, i.e., the images for both eyes, are rendered sequentially first for one user, then the other, whereas using the eye-interleaved mode the images for one eye of both users are rendered sequentially, before the images for the other eye of both users are displayed. However, the main drawback of this approach is that the refresh rate is cut in half for each user compared to the single viewer setup, resulting in a slight but noticeable flicker effect ([ABF$^+$97]). In [FHH$^+$05] this idea has been extended to a multi-viewer projection-based display system, which enables a maximum number of four users to see perspective-correct stereoscopic images in a comfortable way. These images are displayed in a similar way as in the two-user setup, but using one or two projectors for each user. The number of projectors depends on whether a passive or active setup is chosen. Hence this system is hard to scale to allow more than four users to collaborate because for every new user at least one additional projector is required.

Other workbench- and tabletop-based approaches are the *Virtual Showcase* ([BFSE01]), the *Lumisight table* ([MIO$^+$04]) and the *IllusionHole* ([KKYK01]), which enable also about four users to perceive perspective-correct images. In all approaches different areas on a horizontally mounted projection table can be observed via mirror-based setups ([BFSE01]) or physical view barriers ([MIO$^+$04, KKYK01]), which are attached to the projection screen. The drawback of this approach is the necessary reconfiguration and calibration, when the mode is switched between the multi-user and the regular mode.

A software-based approach of Naemeru ([NKH98]) presents a 3-wall CAVE setup to improve the stereoscopic image for non-head-tracked users. Assuming that the head-tracked user focusses on one side of the wall permanently, the remaining parts of the virtual scene projected on the other walls are deskewed to provide an approximate perspective-correct image to the other users. The drawback is that although the stereoscopic images for the non-head-tracked users are improved, perspective distorted scene content still persists.

Another software-based approach ([SS05]) uses a shared stereoscopic image, where the user-dependent content, e.g., virtual input devices, is rendered for each user separately but in one shared view. Thus every participant of the collaboration notices the input devices of other collaborators, but only the user's individual virtual input device is rendered as seen from the user's viewpoint. The drawback using this approach is that it has to be determined which virtual objects belong to whose individual's view and which scene content is rendered for all users equally. Furthermore perspective distorted scene content still persists.

Thus there exist no approach providing several users view-dependent perspective-correct stereoscopic images, which enables comfortable teamwork in a single projection-based VR system environment and does not require auxiliary hardware or reconfiguration of already existing setups.

5.1.2 Projection-Based Collaboration Concepts

In this section software-based approaches are described that enable several head-tracked users to explore datasets together by providing perspective-correct stereoscopic images without the need to switch glasses or to reconfigure the tracking system. Furthermore no construction of expensive multi-user hardware setups is required.

After a registration process, e.g., indicated by a gesture, teamwork can be performed consecutively, or alternatively in parts of one shared projection screen simultaneously. The introduced approaches incorporate the following benefits:

- enabling teamwork in projection-based VR system environments,
- fitting into already existing VR system setups without the need of additional hardware,
- easy up- and downscaling, i.e., software-based addition resp. removal of teamworkers, and
- enabling communication and face-to-face interaction between participants of the CVE.

Since the objective of this approach is to exploit already existing VR hardware to provide an environment for VR-based teamwork, resources have to be shared. Usually in a projection-based VR system environment there is only one projection screen available for several users. Thus to fulfill the demands of projection-based CVEs, assignment of projection space to participants has to be organized. Allowing several users to participate in the teamwork process requires that their view positions and directions have to be tracked. Furthermore these users should have appropriate, tracked input devices for six DoF interaction. Users who currently participate in the collaboration are called *active users*, whereas users who only observe an interaction process and who perceive distorted stereo images are called *non-active users.*

In this software-based approach for a CVE, there are two different interaction modes for co-located interactions:

1. the *cooperation mode* and
2. the *split-screen collaboration mode.*

Interactive switching between these two modes is possible at runtime. Both the cooperation mode and the split-screen collaboration mode as well as the aforementioned registration process are explained in the next subsections.

Cooperation Mode

In some CVEs it is sufficient to accomplish teamwork in a cooperative mode, i.e., several cooperators interact one after another. When using this cooperation mode only one active cooperator perceives a perspective-correctly rendered image and manipulates the VE, simultaneous collaborations are not supported. Starting with this single-user mode the active cooperator can explore the VE immersively and interact with it until a new user volunteers for cooperation. In a standard projection-based VR system environment, the active cooperator and the new user have to switch their glasses as well as input devices. Afterwards the new active cooperator is able to interact until another user wants to cooperate.

The approach presented in [SRH06, SHR06, SRH05b] enables the following alternative. A new user, who wants to cooperate, can simply volunteer for the cooperation by communicating or satisfying predefined conditions, e.g., posing a special gesture. When the registration is confirmed, e.g., the new active cooperator agrees, the new user will change the status from a non-active user who only observes the interaction to the active cooperator. Now, the position and orientation of the new active cooperator's head and input devices are evaluated to handle the interaction, and the previously active cooperator shares the view frustum of the currently active cooperator.

Indeed, using this approach only one cooperator is active at any time, but the seamless switchover of active cooperators enables groups of participants to cooperate, because no switching of glasses or input devices, or reconfigurations of the tracking system are required as it is necessary in common co-located projection-based VR systems. This mode is favorable in VR systems in which only one user needs to interact at any moment. The active user is aware of the full viewport space. In comparison to both hardware-based solutions for two ([ABF+97]) resp. four ([FHH+05]) users, no flickering occurs since the frame rate is not affected by the number of cooperators. However, every non-active user perceives perspectively distorted scenes. Thus this mode is beneficial for virtual scenes in which the perspective distortion is minimal, i.e., stereo images are projected with nearly zero parallax resulting in a smaller stereoscopic effect (see Chapter 3).

Figure 5.2 shows two cooperators in front of a responsive workbench environment interacting with a virtual city model. The displayed objects are projected stereoscopically such that they appear in a small area above the screen surface. Thus the perspective distortion perceived by non-active users is small, and a change of the active user can be performed with less jerky leaps, which occur when a different user's head position is used for the view-dependent rendering.

Split-Screen Collaboration Mode

Although the cooperation mode enhances cooperative interaction in projection-based VR system environments it does not support simultaneously performed collaboration. For this purpose, another approach proposed in this section exploits the usually large displays provided in projection-based VR environments ([SRH05b]). When using the split-screen collaboration mode the screen is split into appropriately sized viewports arranged side-by-side, in which each active collaborator

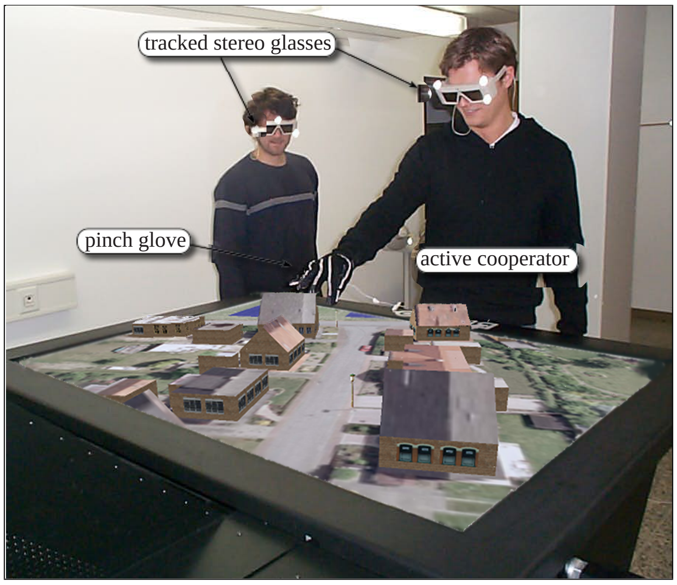


Figure 5.2: Two users in a cooperative interaction mode in front of a responsive workbench.

perceives a perspective-correct image. When additional active collaborators are involved in the collaboration process, the current viewports are split in smaller viewports again. In the general case in which n active collaborators interact in front of a projection screen which is w *inch* wide and h *inch* high, each active collaborator is assigned a vertical area of size w/n *inch* $\times$ h *inch*. When another user registers for collaboration the new areas, which are reassigned to each user, have a size of $w/(n+1)$ *inch* $\times$ h *inch*. In these vertical areas, each active collaborator perceives a perspective-correct stereoscopic image. In the case of removing an active collaborator the viewport areas are scaled accordingly.

Since users collaborate in different working areas and only one tracking system is used to determine the position and orientation of the users' heads and input devices transformation of the tracking data is required. Figure 5.3 illustrates this issue. The tracking coordinate system differs from the coordinate system of the individual viewports, and thus the received tracking data has to be transformed from the tracking coordinate system to each active collaborator's viewport coordinate system. The coordinate system of the tracking system and the coordinate system of each active collaborator's viewport have the same orientation, and they can be transformed into each other by applying an appropriate translation.

In the case of the vertical arrangement the transformation of the tracking data to the viewport number i is done by applying a translation vector $t_i^T = (-\frac{w}{2} + (i-1)\frac{w}{n} + \frac{1}{2}\frac{w}{n}, 0, 0)$ to the data obtained from the tracking system, where again w is the width of the projection screen and n denotes the number of collaborators.

Although active collaborators can see the viewports of other active collaborators, a perspective-correct image cannot be observed within these, and thus interactions of each collaborator are constrained to the corresponding individual viewport. However, manipulations of other collaborators can be observed comfortably in the individual viewports while communication as well as

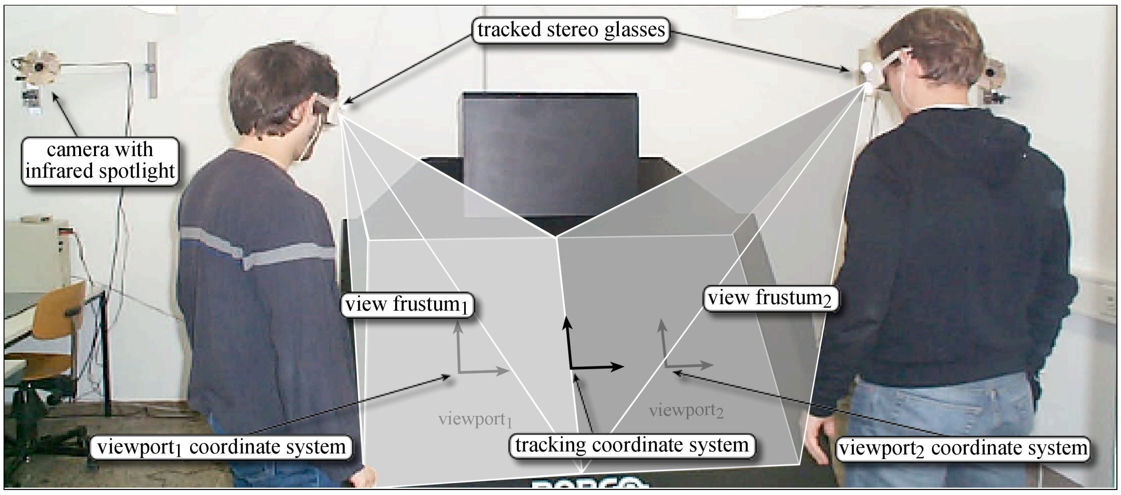

tracked stereo glasses
camera with infrared spotlight
view frustum$_1$
view frustum$_2$
viewport$_1$
viewport$_2$
viewport$_1$ coordinate system
tracking coordinate system
viewport$_2$ coordinate system

Figure 5.3: Two users collaborate in the split-screen collaboration mode in front of a responsive workbench.

face-to-face collaboration is ensured. Therefore no avatars are required as in distributed CVEs (e.g., [LJB+99]).

Figure 5.3 shows two active collaborators interacting in front of a shared projection screen in a responsive workbench environment. The scene is rendered twice; in the left resp. right viewport it is displayed for the left resp. right collaborator with the corresponding camera settings. Each collaborator interacts within his individual viewport. Position and orientation of the heads as well as the input devices are tracked by an optical tracking system. The data received from the tracking system is transformed to the corresponding coordinate system of each user as described above. The individual asymmetric view frustum of each collaborator is calculated according to the head position and the corresponding viewport on the projection screen. The view frustums are down- resp. upscaled, when another active user participates in the interaction or an active user quits the collaboration.

As described above the shared projection screen can be tiled vertically into viewports and thus users can collaborate side-by-side. Alternatively the screen can be tiled horizontally or in a quadratic manner. The drawback of using non vertical tiling is that conflicts in front of the projection screen are possible, since users may collide because their working areas overlap partially. Indeed using this approach the original size of the screen is downscaled; but since large projection screens provide enough space the split-screen collaboration mode enables several users to collaborate in projection-based VR systems, while each collaborator still has a sufficiently large individual viewport at his disposal. The number of possible active collaborators in such a setup depends on the size of the projection screen and the flexibility required for each user. For example, with horizontally tiled viewports the maximum number of active collaborators depends on the size of the projection screen and the interactivity required by the application. In a responsive workbench system environment, best results have been experienced with up to three collaborators working side-by-side.

Figure 5.4 shows two users in a responsive workbench setup collaborating side-by-side in (a) a

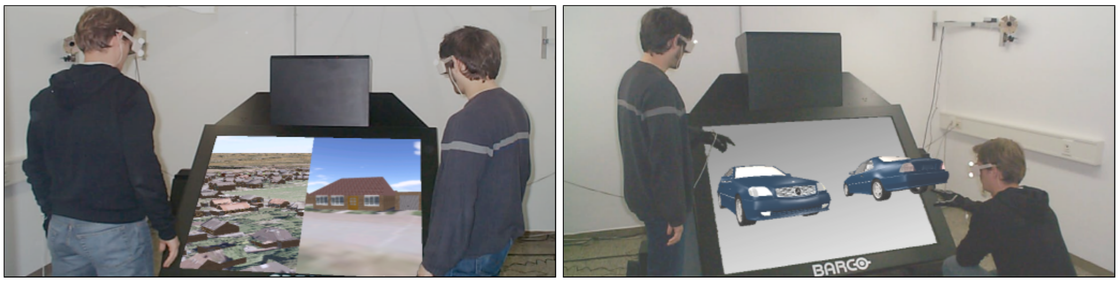


Figure 5.4: Two prototype applications for the split-screen interaction mode: (a) residential city planner (see Chaper 6), (b) automobile viewer.

3D residential city planning environment and in (b) a virtual automobile exploration application (right). The active user, standing on the left side of the responsive workbench, perceives a perspective-correctly rendered stereoscopic image in the left viewport. The scene for the second active user on the right side of the workbench is projected onto the right viewport and appears perspectively distorted to the left user.

In addition, combinations of the cooperative interaction mode and the split-screen collaboration mode are possible to prevent too small viewports. The split-screen collaboration mode allows collaborators to interact simultaneously in an appropriate number of separated viewports each with a certain number of cooperators who perform cooperative interaction in their viewports. Thus, more users can participate in the CVE without the need of scaling the viewports' sizes.

Registration Process

Since in projection-based VR systems there are confined resources, e.g., tracked stereo glasses or viewport space, these resources have to be shared appropriately. To organize the demands of resource sharing teamwork should be administered by the VR software system depending on the specific CVE and the application. As mentioned before, users have to register to participate in the teamwork process. To organize the administration of many participants in a CVE a hierarchical authorization structure similar to the structure used in operating systems should be implemented. The types of access rights can be defined by an administrator according to profiles associated with the different participants, e.g., higher privileged users can allow or deny teamwork with lower privileged users. However, since usually in a projection-based CVE teamwork is performed between a small number of users, sharing of the resources can be realized by assigning equal rights to all users and a registration processes.

For the registration three different strategies are proposed in ([SRH05b]), called *announcement*, *invitation* and *time-dependent switch*. These three approaches are explained in the following.

Announcement

To announce for teamwork a user can perform predefined actions, e.g., by using speech control or posing a gesture, which indicates that the user wants to participate. If the user has appropriate rights, teamwork can be initiated immediately or after a privileged user accepts the announcement. This approach is favorable in CVEs with a small number of collaborators or cooperators.

Invitation

Alternatively, a user, e.g., an active collaborator with sufficient rights, can invite another user for teamwork. The invitation can be indicated, for example, by a gesture after communication between the users. When the invited user accepts the invitation, e.g., again indicated by a gesture, the user's status is switched from a non-active user only observing the interaction to the active cooperator or to an additional active collaborator.

Time-dependent Switch

In some CVEs another strategy might be favorable. When one or more users interact together in a CVE, after a certain time period the active cooperators resp. active collaborators are automatically switched. This time-dependent exchange of active users may be useful in presentation scenarios where the exploration of virtual datasets by groups of users has to be ensured.

Multimodal Registration Process

To reduce the cognitive effort of involved users the number of required gestures used for the registration process should be small. Besides the time-dependent switch, where a change of the active users takes place automatically, the announcement and invitation registration process can be performed, for example, by one of the following gestures. A *notify-gesture*, e.g., consisting of a combination of a glove event (pinching thumb and index finger) and a corresponding tracking event (hand is higher than the head), indicates an announcement for a participation in the interaction process. Afterwards current active collaborators can accept this announcement with the *confirmation-gesture*, e.g., posing a circle by pinching thumb and index finger.

After a successful announcement for teamwork the actions described above for exchange of active cooperators resp. addition or removal of active collaborators are initiated; in the cooperative interaction mode the active cooperators are exchanged. The new active cooperator remains active until the system receives another registration announcement, which has to be confirmed first.

Although the users can see each other, application of the gesture-based registration processes has shown that users often do not observe each other when they perform gestures. This is due to the fact that the active user usually concentrates on the interaction, and non-active users stay beside or behind the active one. To draw the attention of the active user to a cooperator or a collaborator, who wants to participate in the interaction, multimodal interaction concepts have

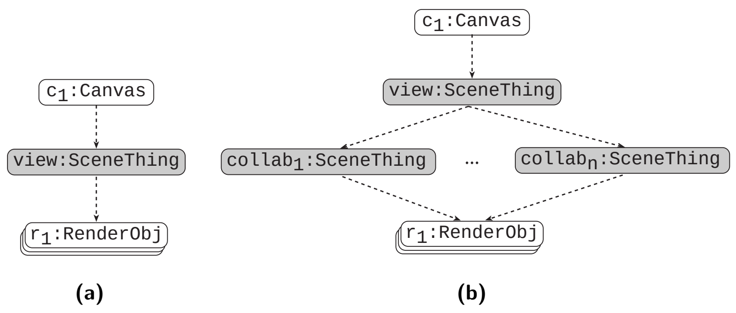


Figure 5.5: Scenegraph (a) before and (b) during collaboration between n collaborators.

been integrated in the registration process. For example, when a user performs an announcement, a spatial sound disperses from the position of this user. Thus, the position-dependent acoustic dispersion gives a hint about which user wants to volunteer for interaction.

Also further multimodal concepts such as haptic feedback as a hint for the indication of a registration process are possible, if the users' input devices feature appropriate technology (see Chapter 3). In addition, the registration process can be indicated by arbitrary multimodal events caused, for example, by speech commands.

5.1.3 Integration of Co-located Interaction Concepts into VR^{2}S

The class `Collaboration` enables teamwork in a CVE. An object of this class as part of the behavior graph manages the registration process as well as the realizations of the CVE, i.e., predefined events are handled, which cause corresponding callbacks. The resulting callbacks handle the teamwork interaction processes, e.g., they determine whose head is tracked for view frustum calculation and which input devices are used for manipulation tasks as described in Section 5.1.2. The combination of event and corresponding callback provides a flexible mechanism is provided to change the method how the registration process is performed. An example setup consists of the described gesture-based registration process. For this purpose pinch gloves and an optical tracking system are used to generate the described events. Alternatively, active users can be switched by key events allowing an administrator to control the CVE from outside the VR system environment.

Every CVE encompasses data to be *shared* by all collaborators as well as *individual* scene content which should be visible in the individual viewport of an active collaborator only, e.g., VR widgets supporting the collaborator during manipulation tasks. The distinction between user-specific and shared data is ensured using the proposed approach, i.e., each scene node can be specified as individual or shared data. The `Collaboration` class handles a corresponding rearrangement of the scenegraph such that all collaborators are able to see the shared as well as their user-specific data. Thus, in every viewport the shared content plus the user-specific data are visible in a perspective-correct image. As long as collaborators manipulate shared scene content, manipulations are visible in the viewport of every other collaborator, whereas manipulations of scene content which is user-specific remain invisible to the other collaborators.

Figure 5.5 shows an example of the scenegraph rearrangement during an initiated collaboration. In Figure 5.5 (a) a simple scenegraph is appended to the canvas. The scene node `view` contains basic scene information. Several instances of type `RenderObj`, which represent graphics objects and attributes, are appended to this scene node, and therefore visible to all active collaborators during collaboration. The rearranged scenegraph resulting from a collaboration between n users is illustrated in Figure 5.5 (b). Each of the n nodes, $\texttt{collab}_1$,...,$\texttt{collab}_n$, contains viewport and transformation information, which enables rendering of perspective-correct graphics objects in the corresponding viewport. In addition these nodes can contain user-specific data that is only visible in the individual viewport of each user.

5.1.4 Preliminary Usability Study

To evaluate the concepts described above a preliminary usability study has been performed in which the proposed approaches for cooperative and collaborative as well as single-user interaction have been evaluated. Since the results of this study should give a first impression of the acceptance and applicability of the described concepts, 10 participants have been asked to assess the approaches. The participants were VR-novices such as students of computer science (3), mathematics (1) and geoinformatics (3) as well as research assistants (3), who were already familar with VR technologies.

Tasks

For the evaluation of the concepts, the applications depicted in Figure 5.4 were used in a responsive workbench environment. The virtual scenes have been presented to each pair of participants, one of whom wore tracked stereo glasses, whereas the other wore non-tracked stereo glasses. To change the active user, the participants had to switch the glasses. In the second phase, both participants wore tracked glasses, and the cooperative interaction mode was tested with the described registration processes, initiated by gestures. For each approach, i.e., announcement, invitation and time-dependent switch, a short introduction into the functionality of the techniques was given. In the last study phase the split-screen collaboration mode was used to find out in how far the visualization on a tiled projection screen affects the subjective perception of the participants. For the evaluation of the usability study, the participants were asked to review the different interaction modes and approaches for the registration process. Most questions were based on a five-point Likert scale (from 1 to 5 associated with corresponding ratings).

Results

The objective of this user study was to evaluate the usefulness of the described concepts and to identify drawbacks and opportunities. Although the task was constrained to an exploration exercise, the results show the potential of the described strategies.

The manual switching of tracked glasses and input devices took at least five seconds and as expected participants felt inconvenient about it. They rather prefer to watch a virtual scene without tracked glasses than to switch the glasses again and again.

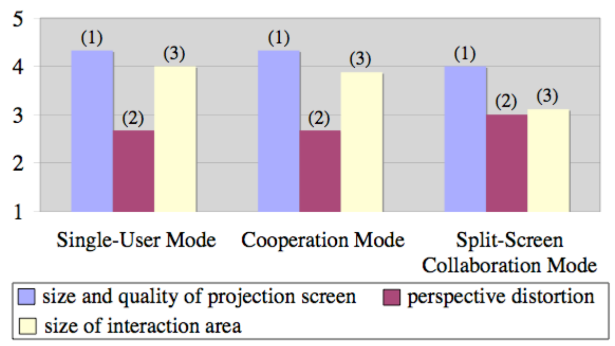


Figure 5.6: Results of the user survey.

As depicted in Figure 5.6 the review of the quality and size of the visual representation in the two-user split-screen collaboration mode shows that these qualities do not decrease significantly, and distraction by perception of other users' viewports was rated as minor (5 corresponds to sufficient size and quality of projection screen (1), no disturbing perspective distortion (2) and sufficient size of interaction area (3), whereas 1 corresponds to insufficient size and quality of projection screen (1), very disturbing perspective distortion (2) and insufficient size of interaction area (3)). The interaction space in front of the RWB was rated as sufficient, but a larger interaction area would be preferable especially in the split-screen collaboration mode and for more than three participants.

The complexity and usability of the proposed gesture-based registration process was rated with 4.8 on average (1 corresponds to very complex and not intuitive, 5 corresponds to not complex and very intuitive), and the support by multimodal feedback was considered as very helpful (on average 4.1 where 1 corresponds to not helpful, 5 corresponds to very helpful).

5.2 Advanced Mixed Reality System Environments

This section introduces concepts for virtual global illumination ([SHR05]). To enhance immersion in VE, virtual objects are augmented with real world information regarding the VR system environment. Real-world objects such as input devices or light sources as well as the position and pose of the user are used to simulate global illumination phenomena, e.g., users can see their own reflections and shadows on virtual objects. Besides the concepts and the implementation of this approach, the system setup and an example application are discussed.

5.2.1 Mixed Reality Environments

Seamlessly merging the real world with the virtual world created within a computer is a challenging topic in current VR research. Technology that superimposes the real world by computer-generated images is called *augmented reality (AR)*, whereas the enhancement of virtual worlds using real-world data is called *augmented virtuality (AV)*. The term *mixed reality (MR)* ([MK94]), encompasses both augmented reality as well as augmented virtuality.

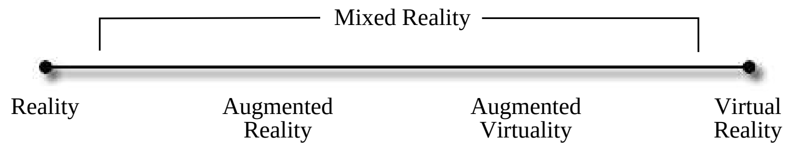


Figure 5.7: Reality-Virtuality Continuum (adapted by [MK94]).

Figure 5.7 depicts the transitions between the different kinds of immersion in the so-called *Reality-Virtuality Continuum.* As mentioned in [MDG$^+$95], the main issues of MR environments are consistency of geometry, time, and illumination between real and virtual world. Of course, one of the most important tasks is that superimposed objects have to be placed at the exact position where they would exist in the real world. Likewise, reflections, light sources and shadows must match in both worlds to obtain consistent global illumination; hence the real and virtual world must be synchronized.

5.2.2 Global Illumination Concepts for MR Environments

In the context of MR environments, some work has been done to simulate global illumination phenomena in order to merge real and virtual worlds. These reproductions of global illumination phenomena focus on the generation of reflections and shadows.

Surfaces such as glass, metal or water reflect their environments. To reproduce this effect in computer graphics, *environment mapping* has been introduced by Blinn and Newell in 1976 ([BN76]), which can be used to approximate reflections of arbitrary shaped objects. An environment map is a texture containing information about the surrounding of a virtual object. During the rendering process corresponding texture coordinates are applied to virtual objects, which reflect the environment map. For this purpose Green ([Gre86]) has proposed *spherical environment maps* and *cubic environment maps.*

In a spherical environment map a sphere with a single spherically distorted texture of the surrounding encloses the virtual scene. In the case of a cubic environment map the virtual environment is approximated by six faces of a cube having an appropriate texture map; the cube is centered at the camera position. To simulate reflections rays are cast from the virtual camera's position to the objects and they are reflected through the environment map. The filtered color at the point of intersection with the virtual sphere or one of the six texture images is used for the reflection on the corresponding surface point of the object. Green ([Gre86]) preferred the usage of cubic environment maps because of the easier integration into 3D graphics hardware. Cubic environment maps are created by rendering a virtual scene or capturing real-world information with a 90-degree FoV camera resulting in *left*, *right*, *front*, *back*, *top* and *bottom* textures. Nowadays environment mapping is accelerated by graphics hardware and is used in computer games and interactive applications to simulate reflections while preserving high frame rates.

In the area of simulating global illuminations, most work is based on the reproduction of shadows in the virtual environment ([Wil78]). Most of the used *shadow algorithms* and methods can be divided into two main categories – *hard* and *soft shadows* – depending on the type of shadow that they produce. Hard shadow computation is very close to normal visibility testing; basically it is just a visibility test with respect to a point light source. Soft shadow algorithms generate more realistic looking shadows, however, they require more calculation effort because areas of light sources are considered instead of a single point light source as done when using hard shadow computation. In order to preserve high frame rates when reproducing global illumination phenomena in VEs, this section concentrates on hard shadow algorithms ([Wil78]). Similar to visible-surface algorithms, shadow algorithms determine which surfaces can be seen from the light source; surfaces visible from the position of the light source are not in the shadow, whereas surfaces that are not visible are in shadow. For multiple light sources, surfaces must be examined with respect to each of the light sources.

In the context of global illumination in computer graphics, recent approaches blend both synthetic as well as real objects to generate images of excellent quality. However, global illumination is applicable only under specific limitations, e.g., non real-time performance, or if certain requirements are satisfied, e.g., a geometric computer model of the real scene is available or light sources are static ([JL04]). The strategies proposed in [NNMH02], which improve the usage of virtual lights and shadows in MR, use special hardware that is usually not accessible in most MR system environments. The effect of virtual reflections with respect to immersion in AR environments has been examined in [RWH04]. This approach approximates reflections of real-world objects on virtual objects by extracting environment information from the background.

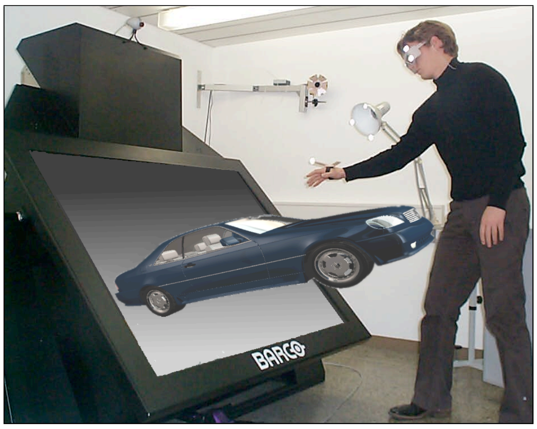
5.2.3 Mixed Reality Environment Setup

To blur the borders between the real and the virtual world visual information about the real environment surrounding the user is added to the virtual objects, e.g., objects dynamically mirror the environment, and thus users can see their own reflections and may also cast shadows in the virtual world. The concepts support the realization of an MR environment that provides a novel way of visual exploration.

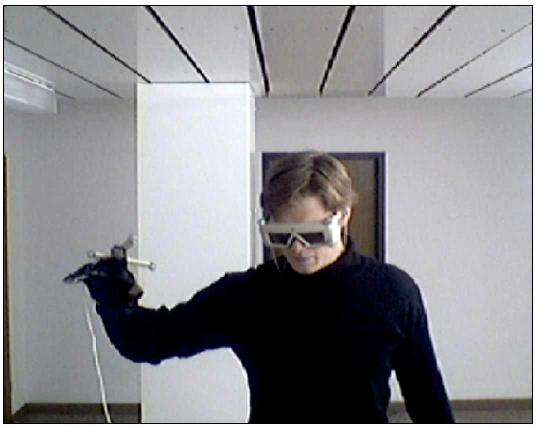
Virtual Reflections

As described in Section 5.2.2, in real-time computer graphics usually an environment map is constructed in order to generate reflections. Since the textures are static images that are not affected when the environment changes, interaction may have unexpected and inconvenient effects in a highly interactive MR environment. This is due to the fact that in contrast to the real world, the movements of the user have no influence on the visualization of the virtual objects. To further improve immersion in interactive MR applications changes of the real-world environment have to be incorporated. Therefore a dynamic environment map representing the complete surrounding of the MR system environment is desirable.

In the approach presented in [SHR05] this idea is simplified. A single USB camera records the main working area of the user, e.g., the area in front of the screen in a workbench-based MR system environment. The remaining areas are given by static images of the surrounding



(a)



(b)

Figure 5.8: (a) MR system setup with responsive workbench, tracking system and USB camera. (b) A grabbed frame from the USB camera.

generated by rendering a geometric model of the MR system environment or by taking photos with a 90^o FoV camera.

An example setup showing this approach is illustrated in Figure 5.8 (a). The camera is attached to the top of a responsive workbench. Figure 5.8 (b) shows a user in the working area captured as a stream.

This image replaces the front texture of the cubic environment map. Figure 5.9 (a) shows such a cube map of the VR laboratory at the University of Münster consisting of left (1), right (2), front (3), back (4), top (5) and back (6) textures. The front texture is exchanged with a grabbed frame of a USB camera (see Figure 5.9 (b)), which is attached to the top of the workbench as illustrated in Figure 5.8 (a).

Virtual Lights and Shadows

Besides virtual reflections the usage of virtual light sources and virtual shadows enhances the realism of virtual objects ([Wil78]). In order to apply and modify direct lighting in MR environments in an intuitive way, the following light interaction of real and virtual objects can be used ([SHR05]). Passive markers are attached to a lamp for real-time tracking via an optical tracking system (see Figure 5.8 (a)). Position and orientation of the tracked lamp are exploited to place the virtual light source accordingly. If a user moves the lamp, the virtual light moves in the virtual scene in the same way as the real-world light source and illuminates the scene.

In the real world a user moving between a light source and an illuminated object casts a shadow on the object. Because the user is usually not defined as geometry in the visualization system, it is difficult to calculate a corresponding shadow. In general, however, the user's head and at least one hand or input device are tracked to allow direct interaction. In the approach described in [SHR05] the virtual scene is augmented with a geometric model approximating the user's pose used for shadow generation. For instance, a pinch glove and its approximated model

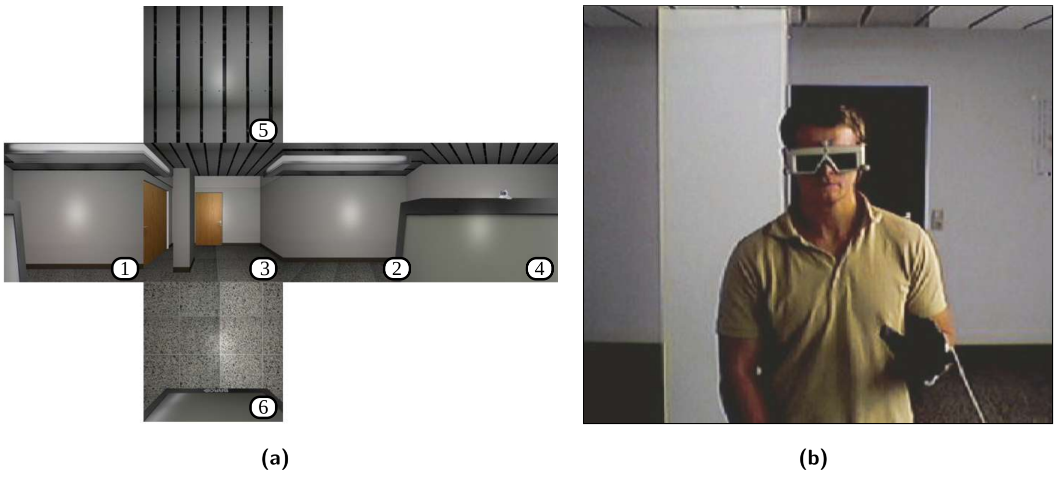


Figure 5.9: (a) Cubic environment map of the synthetically generated model of a responsive workbench of the VR laboratory and (b) the image, grabbed with the USB camera on top of the responsive workbench, to be exchanged with the front texture (3).

in the virtual world are shown in Figure 5.10 (a) resp. (b).

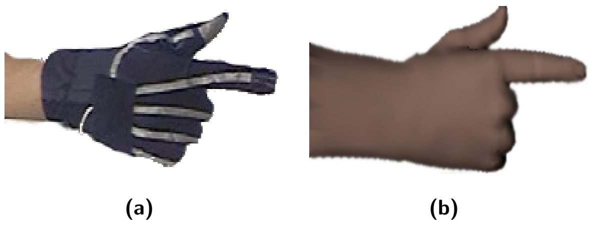


Figure 5.10: Mapping between real and virtual input devices: (a) pinch glove and (b) virtual model.

The depth information of the model's geometry is rendered into a depth texture which is applied later on during shadow mapping. Hence the user, in particular tracked parts of the body, or input devices can drop shadows on virtual objects. Using this approach no further hardware devices are necessary for shadow generation.

Integration of Virtual Global Illumination into VR^{2}S

This section illustrates the integration of the described virtual global illumination concepts, i.e., virtual reflections and virtual shadows, into the VR software system VR^{2}S.

To realize virtual reflections, the ARToolKit ([KB99]) in combination with an USB camera is used. Since the images of the USB camera are grabbed via the ARToolKit interface of VR^{2}S each camera supported by ARToolKit is compatible. To reference ARToolKit and a connected camera, an instance of the class `ARToolKit` has to be appended to the behavior graph. The class `ARToolKit` is derived from the class `Interaction`. Within this class, the camera model is referenced and grabbing is initialized. ARToolKit also provides real-time six DoF tracking of simple predefined markers, which can be exploited for real-time tracking of input devices to which such markers are attached. The method `getVideoFrame` of the `ARToolKit` class returns an object of class `Image`, which can be used to replace an image of the cubic environment map, e.g., the front texture as in Figure 5.9.

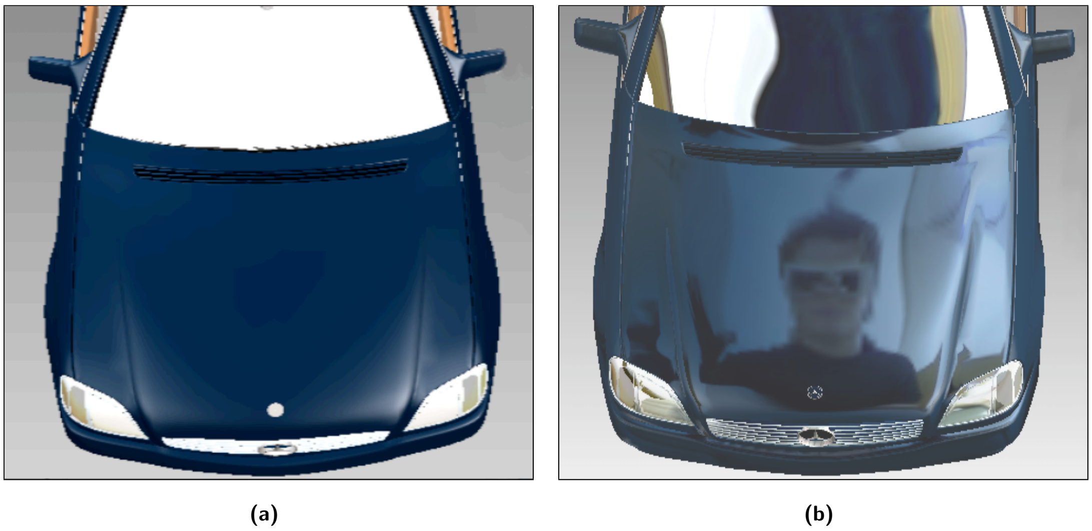


Figure 5.11: A scene containing a virtual car model (a) without and (b) with virtual reflections.

The virtual automobile model in Figure 5.11 (a) shows no reflections, whereas Figure 5.11 (b) shows virtual reflections resulting from application of an environment map containing real-world information. By using this approach, virtual objects with a reflective surface mirror other virtual objects and the static environment as well as the varying working area. Although the lighting conditions of projection-based MR system environments are often insufficient, users experience an adequate reflection because additional light is dispersed from the projection screen.

Listing 5.1 shows the integration of a dynamic cubic environment into VR^{2}S. In line 6 the ARToolKit interface is initialized. Lines 8-19 generate a cubic environment map consisting of the corresponding textures `front.ppm`,...,`back.ppm`. Using the callback method `exchangeTexture` shown in lines 23-25 an arbitrary texture of the cubic environment map is exchanged with a grabbed image from the camera associated with the ARToolKit.

To generate virtual shadows approximations of input devices or certain parts of the user's body are associated with so-called *shadow caster objects*, which throw shadows on other objects.

A part of an example program (see Listing 5.2) shows how virtual light sources can be used in VR^{2}S to enable users to drop shadows into the virtual world. In lines 3-7 three light sources consisting of a point, spot and distant light are generated. In lines 9-13 a `SceneThing` node is defined as object that can drop shadows on other objects, and that can be shadowed by other objects with respect to the distant light source `distantLight`. In lines 16-18 and 20-22 spatial input devices are instantiated and associated with rigid bodies with the unique ID 1 resp. ID 2, assigned to a pinch glove resp. a lamp. In lines 24-26 the tracking system, the input device and the lamp are appended as nodes to the behavior graph.

```
  // global cubic environment map
  SO<Array<SO<Image> > > cubemapImages_
  ...

  // initializing ARToolKit
  SO<ARToolKit> ar_ = new ARToolKit(...);

  cubemapImages_ = new Array<SO<Image> >(6);
  SO<Image> front = VRS_GuardedLoadObject(Image, "front.ppm");
  ...
  SO<Image> bottom = VRS_GuardedLoadObject(Image, "bottom.ppm");
  (*cubemapImages_)[ImageCubeMapTextureGL::Front] = front;
  ...
  (*cubemapImages_)[ImageCubeMapTextureGL::Bottom] = bottom;

  SO<ImageCubeMapTextureGL> texture_ ;
  texture_ = new ImageCubeMapTextureGL(cubemapImages_->newIterator());
  thing_->append(texture_);
  thing_->append(new TexGenGL(TexGenGL::ReflectionMap));
  ...
}
...
void exchangeTexture(Image* image) {
  cubemapImages_ ->setTexture(image,ar_->getVideoFrame());
}
```

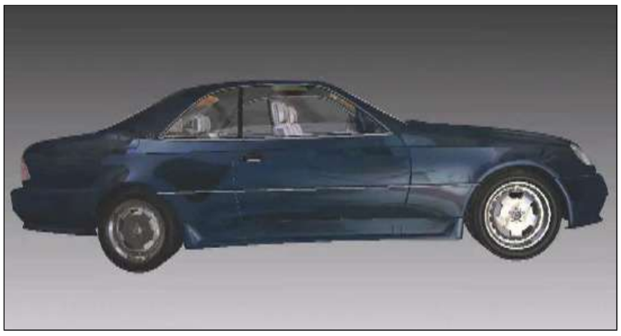
Listing 5.1: Virtual reflection within VR^{2}S.

Both virtual reflections and virtual shadows have been combined in a prototype application that allows VR-based exploration of car models. A virtual car can be illuminated intuitively by positioning a tracked lamp.

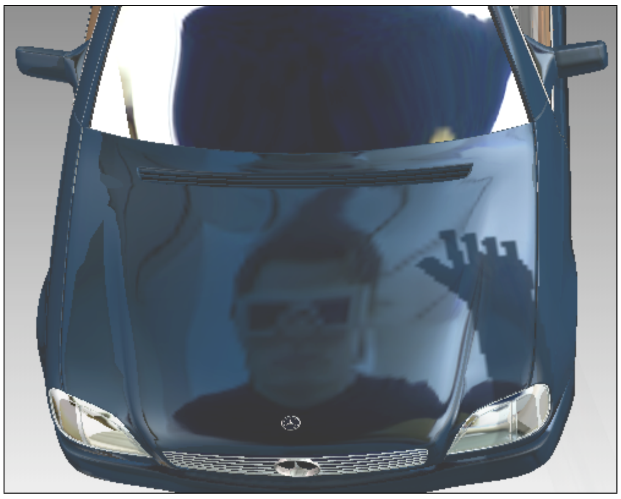
Figure 5.12 (a) shows visible reflections on the auto body, rims and windows as seen from the user's point of view. The reflections from the car surface show the surrounding of the VR laboratory. Figure 5.12 (b) visualizes reflections and shadows on the engine hood.

```
...
// light geometry node
SO<SceneThing> light = new SceneThing(main);
SO<DistantLight> distantLight = new DistantLight());
light->append(new PointLight(Vector(0,0,0)));
light->append(new SpotLight (Vector(0,0,0), Vector(1,0,0)));
light->append(distantLight);
 ...
// input device geometry node
SO<SceneThing> hand = new SceneThing(main);
SO<ShadowCaster> cast = new ShadowCaster(distantLight);
hand->append(cast);
hand->append(new Shadowed(distantlight));
...

//  tracked pinch glove with ID 1
SO<SpatialInputDevice> sid = new SpatialInputDevice(hand)
sid->setRigidBodyID(1);

//  tracked tactile input device with ID 2 to control lamp
SO<SpatialInputDevice> lamp = new SpatialInputDevice(light);
lamp->setRigidBodyID(2);

canvas->append(new TrackingSystem(camera));
canvas->append(lamp);
canvas->append(sid);
...
```

Listing 5.2: Virtual lights and virtual shadows controlled with a lamp and with an object of type `SpatialInputDevice` within VR^{2}S.



(a)



(b)

Figure 5.12: MR system results (a) with virtual reflections and (b) with virtual reflections and virtual shadows.

Chapter 6

Case Studies: Applications for Semi-Immersive Virtual Reality Systems

This chapter presents two example application domains, which benefit from the concepts for generic interaction tasks in virtual reality environments proposed within this thesis. Technical details about two prototype semi-immersive VR system setups are explained in detail.

In recent years *virtual reality based geographic information systems (VRGIS)* have been employed successfully to accomplish GIS-specific tasks ([RHS05, KLR$^+$95, Fau95]). Tracking technologies and stereoscopic visualization of three-dimensional structures support the user to gain a better insight into complex datasets. Moreover, large projection-based displays have shown considerable potential to enable collaboration in co-located VRGIS setups. However, due to the lack of intuitive interaction concepts, they are used mainly as advanced visualization tools. In order to enhance the interaction process in VRGIS-based environments the metaphors proposed in Chapters 4 and 5 as well as their extensions are applied to two example applications from the GIS domain.

This chapter is structured as follows. Section 6.1 describes the semi-immersive responsive workbench, in particular its technical characteristics. Section 6.2 discusses technical details about a semi-immersive passive rear projection system that has been installed within the scope of this thesis. Section 6.3 describes the *3D residential city planner* application, which enables city planners to design and evaluate virtual 3D city models within both VR system setups such as the responsive workbench as well as desktop-based environments. Section 6.4 gives an overview about strategies for the interactive control of *focus and context* concepts when visualizing seismic volume datasets within VR-based environments. These strategies allow to emphasize certain regions of interest in these datasets. In particular it is discussed how the presented visualization strategies benefit from the usage of the proposed semi-immersive VR systems.

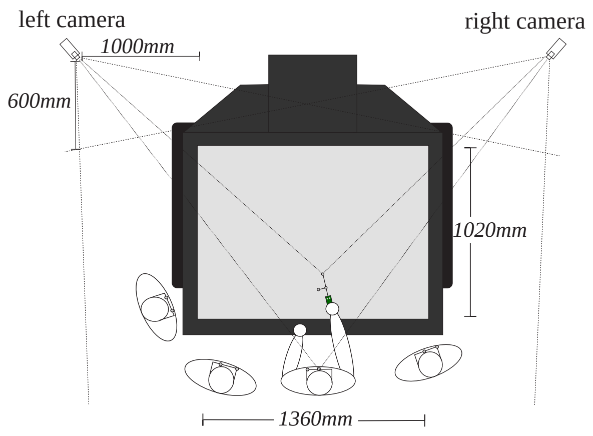


Figure 6.1: Conceptual view of the semi-immersive RWB from the top view. Two cameras track the area in front of the RWB, where four users collaborate (adapted from [DU02]).

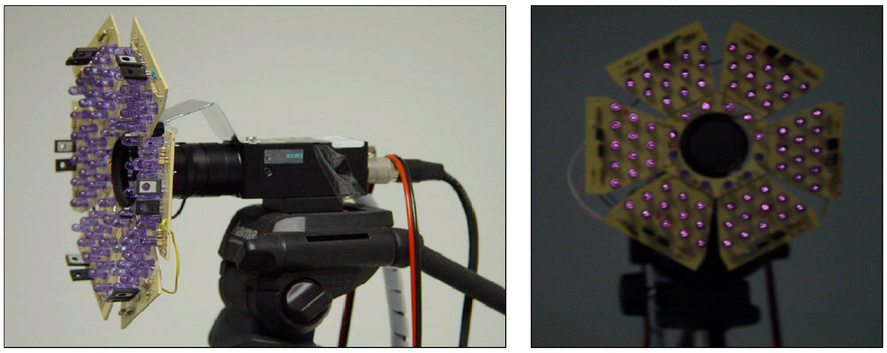
6.1 The Responsive Workbench Environment

For many domains using table-top setups is beneficial because the interaction with data or objects arranged on a table is considered a common task. Objects can be moved across and be aligned with the surface of the table. The responsive workbench is a VR-based projection system, which realizes such a table-top setup.

Responsive workbenches, for example the Barco BARON responsive workbench ([KBF+95]), can be used to project virtual environments stereoscopically to enable users to work in an environment they are accustomed to. A CRT projector contained in the case of the RWB displays images via a mirror onto the back of the transparent projection screen. The projection screen of the workbench can be mounted horizontally or tilted similar to a desk (see Figure 6.1). The virtual scene can be projected onto the display in a way that users perceive virtual objects as being attached to the projection surface, when rendering the VE stereoscopically with zero and negative parallax (see Chapter 3). The BARON is an active stereo rear projection system, i.e., the images are displayed in sequence, first the image for the left and then the image for the right eye. Therefore, the displayed images have to be synchronized with active shutter glasses worn by the users. An emitter sends the corresponding signals to the shutter glasses in sufficiently small intervals. When wearing stereo shutter glasses real-world objects such as the RWB and input devices can be seen and hence users are semi-immersed into the virtual world (see Chapter 2).

The workbench is about $2m \times 2m$ large and $1.2m$ high. The display screen measures $1.36m \times 1.02m$ with a maximum pixel resolution of 1280×1024. The maximum refresh rate of 120Hz is ensured with a resolution of 1024×768 pixels, which allows comfortable working without annoying flickering effects.

As illustrated in Figure 6.1 the size and resolution of the workbench allows several users to view virtual objects in a stereoscopic projection (see Chapter 5). Users can walk around the RWB and view the virtual 3D data from different perspectives. To enable such an exploration from different view positions, the user's current position needs to be tracked in order to determine



(a) Side view **(b)** Front view

Figure 6.2: Cameras attached with IR LED clusters from (a) the side view and (b) from the front view (by courtesy of Dorfmuller-Ulhaas).

the settings of the virtual camera as described in Chapter 3.

A stereo-based optical tracking system consists of two cameras which are arranged as depicted in Figure 6.1. IR LED clusters attached to the cameras (see Figure 6.2) enable tracking of passive markers as discussed in Chapter 3. With such an arrangement an area of about $3m \times 3m \times 2.5m$ can be monitored with an accuracy in the range of millimeters. The tracking system runs on a separate computer and therefore does not affect the graphics performance of VR-based applications. The tracking data is sent via network to the computer running the VR-based application. This network communication is based on the *user datagram protocol (UDP)* which is available for most platforms.

In comparison to front-projection systems the RWB as a back-projection system has the advantage that interferences caused by users standing between the projector and the projection screen are prevented. However, active stereo requires the use of expensive active shutter glasses, which are inconvenient to wear because of their weight. Furthermore, shutter glasses absorb a high amount of light, and synchronization with the sequentially displayed stereo images can cause flickering effects.

6.2 The Passive Rear Projection System

The main drawback of the responsive workbench is the usage of active stereo. In contrast to the responsive workbench, *passive projection systems* do not require active shutter glasses. Since the users wear low-cost passive stereo glasses, these systems are suitable for presentations involving several users. Moreover, the light weight of these passive glasses makes them comfortable to wear even for a longer period of time.

Most passive stereoscopic projection systems use two projectors, which are mounted on top of each other or side-by-side. One projector displays the image for the user's left the other the image for the right eye. For this purpose, different projector technologies are available. While CRT projectors are often hard to calibrate and cause flickering effects and LCD projectors

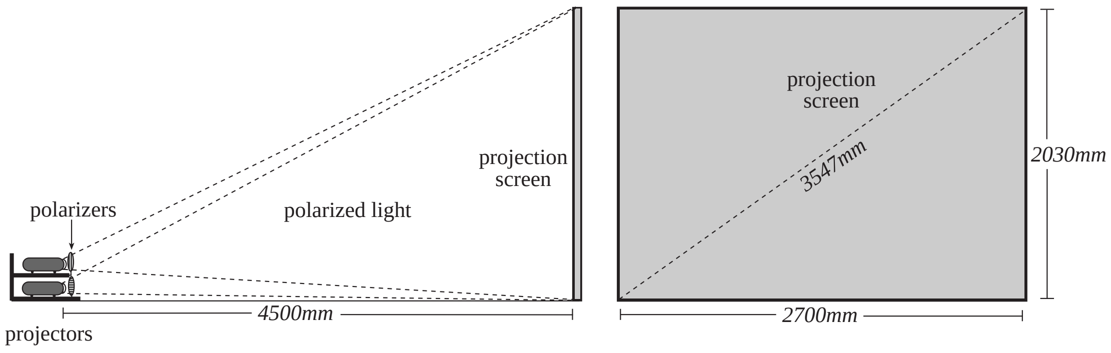


Figure 6.3: Conceptual view of a semi-immersive rear projection system from the side view (left) and the semi-transparent projection screen from the front view (right).

are problematic because their light is often partially polarized, DLP projectors are suitable for stereographics projection since they produce non-polarized and bright light. Non-polarized light is needed since both images displayed by the projectors have to be polarized by linear or circular filters attached to the front of the projectors. If the projection screen preserves the polarization, matching filters in passive stereo glasses can separate the two images for the left and the right eye such that the user perceives a stereoscopic image. The main benefit of circular polarizers is that the user can rotate his head without losing the stereoscopic effect, whereas when using linear filters the separation of the images may be disturbed when the filters are not aligned correspondingly.

The stereo images can be projected onto the screen from the back. As mentioned before problems arise when using front projection setups since users interacting in front of the projection screen may drop shadows onto the screen. Hence, VR systems usually use rear projection which needs additional space behind the projection screen.

In the following subsection the prototype passive rear projection system in the VR laboratory of the Department of Computer Science at the University of Münster is described. This semi-immersive VR system has been built to overcome the drawbacks of the previously described RWB environment.

As illustrated in Figure 6.3 the system consists of two DLP projectors in combination with linear polarizers. The distance between the projectors and the semi-transparent projection screen is about $4.5m$. The frame of the projection screen is $2.73m \times 2.06m$ large, the display screen itself measures $2.700m \times 2.030m \times 0.005m$ with a native pixel resolution of 1024×768 resulting in an aspect ration of $4:3$.

Size and resolution of the projection screen enable several users to view VEs stereoscopically. Because of the rear projection technology users can walk freely in front of the projection screen without dropping any shadows onto the screen surface. Similar to the RWB system setup, an optical tracking system enables view-dependent stereographic rendering (see Chapter 3).

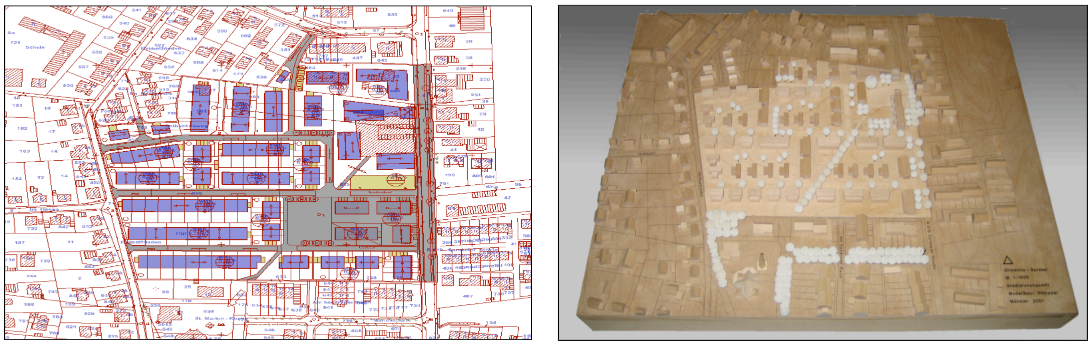


Figure 6.4: Example section of an urban development plan embedded into cadastral information (left) and a corresponding physical block model made of wood downscaled to 1:1000 (right).

6.3 3D Residential City Planning Environment

Urban planning tasks are of major importance for civil works since both the environment and the inhabitants of a city are affected. The cityscape as well as the quality of life of the residents depend on the appearance of buildings, road networks, planting, green spaces, and recreation areas such as parks and playgrounds. To facilitate a realistic impression of how a building area would visually integrate into the environment and to enable communication about different development proposals, it is important that intermediate results of the planning process are presented in a comprehensible way.

By using two-dimensional CAD concepts urban planners design *development plans* for a certain area based on existing *cadastral data* available for every town in Germany. As depicted in Figure 6.4 (left) cadastral data usually specify building footprints, number of floors and floor's height for each building, parcel boundaries, and other information. Development plans define several geospecific entities, e.g., building and recreation areas, associated with a set of constraints, which specify what types of geoobjects are allowed and what constraints have to be considered. When a development plan exists urban planners can propose and design building developments for the area of the plan. The resulting *design plan* illustrates, for instance, potential home buyers how the residential area will look like. After urban planners have agreed to a certain design plan, two different procedures are commonly used.

One approach is to deliver the design plan to an architectural office. On the basis of this plan architects generate virtual 3D models and return exemplary 3D visualizations of planned areas to the urban planners. This procedure has the following two major disadvantages. The returned visualizations are static, therefore urban planners cannot explore the 3D models interactively. Furthermore, modifications to the 3D models proposed after reviewing the 3D visualization cannot be performed by urban planners. Instead, the architects have to be involved again to modify the 3D model. During a planning task, this usually takes several iterations resulting in inefficiency as well as unnecessary expense.

The second common alternative to communicate urban development proposals is to build a physical block model usually made of wood, plastic or paper. Figure 6.4 (right) illustrates such a physical block model for the design plan depicted in Figure 6.4 (left). After a physical block model has been finished, performing modifications is often awkward, since most elements are inflexible and fixated to the model. Furthermore, the creation of these models is a time consuming costly process, which requires a lot of manpower. Thus simpler solutions to visualize planned development areas are desirable.

In cooperation with the urban development, city planning and transport planning office as well as the land surveying and land registry office of the city of Münster in Germany, solutions to these problems have been proposed ([SRHM06]). An objective of this cooperation is to develop computer-aided concepts, which serve the needs of professional urban planners and provide a convenient alternative to current planning tasks. Urban planners demand that the developed concepts should be based on their current processes which result in physical block models as well as computer generated 3D visualizations. However, the planners wish to have more independent and sophisticated control over both approaches; they want to be able to generate virtual 3D block models and 3D visualizations autonomously. Furthermore the intuitive comprehension when viewing a physical block model should be preserved.

The so-called *3D residential city planning* application ([SHR06, SRHM06]) enables urban planners to create proposals for design plans that can be modified and explored in both desktop-based as well as VR-based environments. This application has been used to perform parts of the usability studies described in Chapter 4 and 5. In the following sections the architecture and main components of this system are introduced.

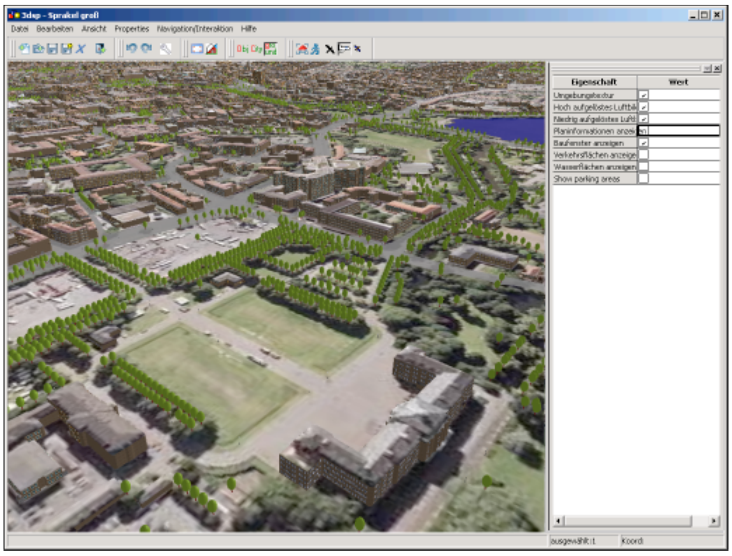
6.3.1 Main Components

Since professional urban planners desire to obtain an intuitive comprehension as perceived when viewing a physical block model, the semi-immersive VR systems in combination with optical tracking systems as described in Sections 6.1 and 6.2 have been chosen to visualize virtual 3D city models. In contrast to physical block models, the usage of such VR systems enables interactive modifications of potential building plans, e.g., changing building parameters such as position or height.

During the development phase of the application, city planners expressed their desire for flexibility when visualizing virtual 3D city models. Although photorealistic rendering is important, *non-photorealistic rendering (NPR)* supports the comprehension of structures and relations similar to view physical block models. During exploration interactive frame rates are more important than photorealistic appearance. However, also realistic visualizations similar to the renderings provided by architectural offices are desired. The 3D residential city planner is based on the VR software system VR^{2}S which supports photorealistic renderers as well as real-time renderers.

The 3D residential city planner and its GUI (see Figure 6.5) consist of four conceptual components:

1. **Converter tool**: The converter tool parses and converts the cadastral data into the underlying geoobject model, which is used to represent the corresponding geodata. Usually,


Datei Bearbeiten Ansicht Properties Navigation/Interaktion Hilfe
ausgewählt:1
Koord:

Eigenschaft	Wert
Umgebungstextur	✓
Hoch aufgelöstes Luftbil	✓
Niedrig aufgelöstes Luftb	✓
Planinformationen anzeig	
Baufenster anzeigen	✓
Verkehrsflächen anzeige	
Wasserflächen anzeigen	
Show parking areas	

Figure 6.5: GUI of the 3D residential city planner.

cadastral data in Germany can be converted into the *drawing interchange format*, sometimes referred to as *drawing exchange format (DXF)*, or into the *shape-file* format.

2. **Geoobject model**: The geoobject model stores the collection of geoobjects and their properties. This model is generated during the parsing process of the converter tool. Components of this model are buildings, building and traffic areas, trees etc. For instance, a building object is composed of a building footprint as well as textured walls, doors, windows, and a textured roof of a certain type.

3. **Visualization component**: This component constructs and evaluates the scene nodes representing the topological structure of the city model as well as information about the environment of the considered city or district. Each scene node in the VRS geometry graph, which represents a collection of geoobjects, is associated with a visual appearance, e.g., by assigning colors or textures.

4. **Interaction component**: This component manages interaction with the entities of virtual 3D city models. The GUI based on *wxWidgets* supports certain desktop-based interactions (see Figure 6.5) as well as direct interaction concepts such as six DoF manipulations of virtual buildings.

Virtual 3D city models can be generated semi-automatically from the geo-referenced cadastral data. Because there is no overall accepted standard for storing cadastral information, an interface provides the required generality and flexibility to enable import of cadastral data from different sources. For instance, for the city of Münster the cadastral data contains several layers storing building footprints and parcel boundaries in *Gauß-Krüger coordinates*, number of floors, and other information, which are converted during the reconstruction process. The interface allows the user to associate semantic descriptions to corresponding layers of the cadastral data.

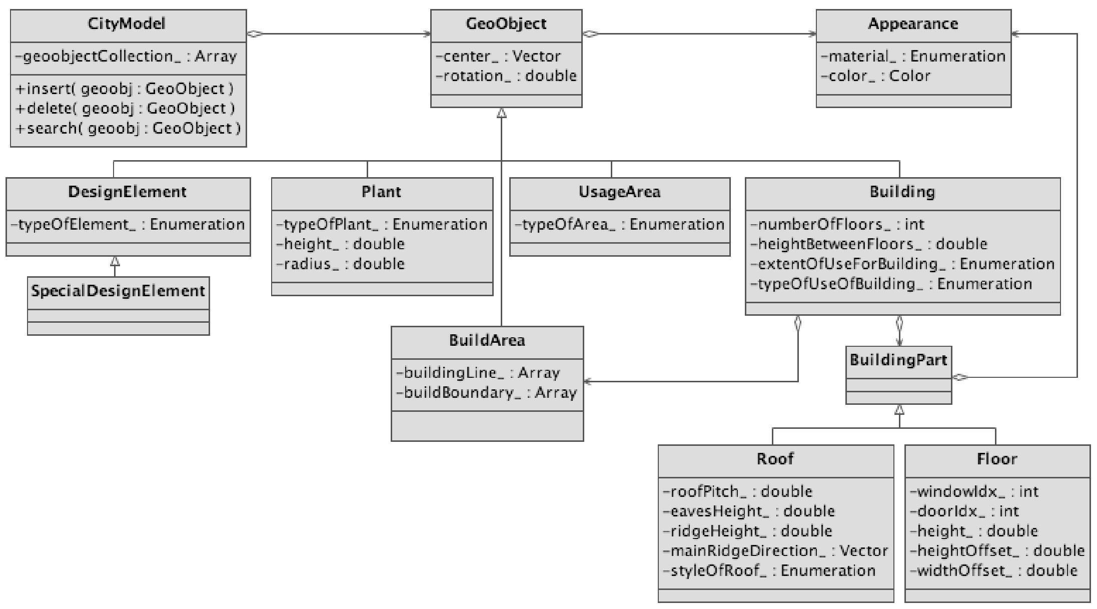


Figure 6.6: UML diagram showing most relevant classes of the 3D residential city planning application.

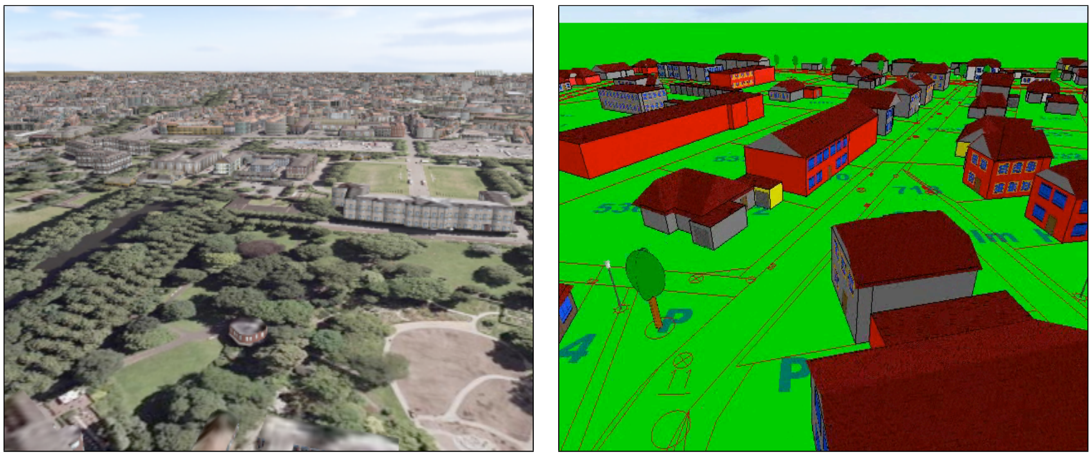
Based on this information the system generates a geo-referenced virtual 3D city model of the surrounding area, which is superimposed with aerial photographs to provide more realism and higher recognition.

6.3.2 Software Architecture

Within the geoobject model all geoobjects are specified by the class `GeoObject`. This class is aggregated in the class `CityModel`, which administrates all required information for a geo-referenced city model (see Figure 6.6). The class `GeoObject` provides the base from which all geoobject classes, for example the classes `Building`, `Plant` or `UsageArea`, inherit. An instance of the class `Building` consists of one or more `BuildingPart` objects to handle different types of storys and roofs. The other geoobjects are organized analogously.

The visualization component is separated from the geoobject model of the virtual city. All required information to visualize the objects of an instance of the class `GeoObject` is handled via the class `AppearanceMapper`. The visual appearance of each geoobject can be assigned randomly, or the city planner can define the appearance, for example by assigning specific textures to each geoobject.

As mentioned above VRS uses a scenegraph to represent virtual environments. Since virtual 3D city models may consist of over 50,000 complex, textured geoobjects, storing each of these geoobjects in a corresponding scenegraph node would inflate memory requirements for storing the scenegraph and decrease performance when evaluating it. Due to the wrapping mechanism of VRS it is possible to store these amounts of data using renderer specific optimization strategies.



(a) Virtual 3D city

(b) NPR city

Figure 6.7: (a) virtual 3D model of the city of Münster and (b) a non-photorealsitic representation with overlaid building plan.

For example, the information about the geometry of all buildings is collected in an OpenGL *vertex buffer*, which is stored and evaluated on the graphics board. Thus it is possible to visualize even large virtual 3D city models at interactive frame rates, e.g., the city of Münster consisting of over 20,000 virtual buildings, trees and further geoobjects is shown in Figure 6.7 (a).

To assist city planners during the design process the graphical representation of both the cadastral plan as well as the building plan can be projected onto the virtual 3D city model (see Figure 6.7 (b)). Performance can be further increased by the optional use of view-dependent level-of-detail algorithms and view frustum culling based on a quadtree representation of the virtual 3D city model. Hence city planners can switch between different levels of realism interactively. Furthermore it is possible to switch between photorealistic (see Figure 6.7 (a)) and non-photorealistic rendering (see Figure 6.7 (b)).

6.3.3 Interaction Concepts for Urban Planning

Besides the visualization of virtual city models, the main task of the 3D residential city planning application is to provide city planners with interaction concepts, which ease the three-dimensional design of development plans. When working in desktop-based environments, professional city planners have to handle 3D interactions with 2D input devices, e.g., the standard mouse or keyboard. To reduce the cognitive effort for such 3D interactions, *3D widgets* ([DH98]) can be used during the manipulation process. 3D widgets provide an easy way to manipulate objects with six DoF by reducing the simultaneously manipulated DoF to one degree. 3D widgets provide handles for translation, rotation and scaling of virtual geoobjects. Thus these manipulation tasks can be performed as easily as in two-dimensions. In addition to 3D widgets which can also be exploited in VR the concepts for VR-based interaction proposed in Chapter 4, e.g., the IVP metaphor, are usable within the 3D residential city planner application as illustrated in

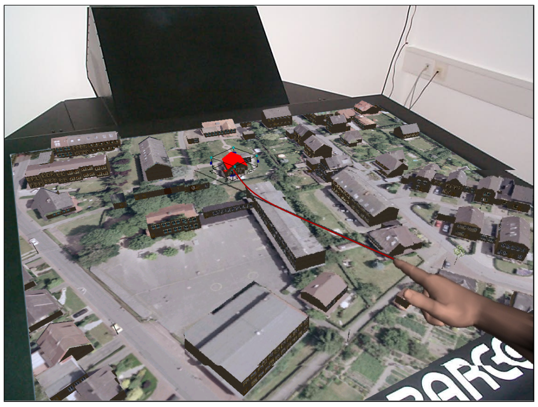
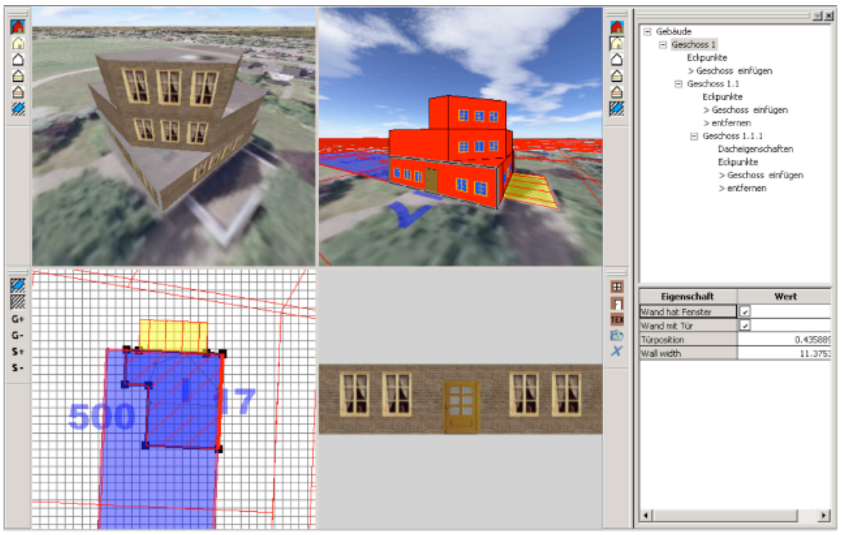


Figure 6.8: Virtual 3D city model and IVP metaphor for interaction with 3D widgets.

Figure 6.8.

Although 3D widgets support the city planner when performing six DoF manipulation tasks, there are many parameters of geoobjects, in particular buildings, which cannot be modified via a 3D widget. For this purpose the 3D residential city planner provides a *building editor*. Using the editor new buildings can be created and relevant properties of existing ones can be modified.



- Gebäude
  - Geschoss 1
    - Eckpunkte
    - > Geschoss einfügen
    - Geschoss 1.1
      - Eckpunkte
      - > Geschoss einfügen
      - > entfernen
      - Geschoss 1.1.1
        - Dacheigenschaften
        - Eckpunkte
        - > Geschoss einfügen
        - > entfernen

Eigenschaft	Wert
Wand hat Fenster	✓
Wand mit Tür	✓
Türposition	0.435889
Wall width	11.3753

Figure 6.9: The building editor is composed of six views: contextual 3D preview (upper-left), 2D vector-based topview (lower-left), lateral view (upper-middle), orthogonal wall view (lower-middle), hierarchical structure view (upper-right), and property view (lower-right).

Figure 6.9 shows the graphical user interface of the building editor. The editor is a *multi-view system* ([SRHB06, NS97]) composed of six views. In the upper-left a *contextual 3D preview* of the virtual building shows its integration into the environment that can be visualized in a semi-transparent way. The lower-left *vector-based topview* shows a two-dimensional presentation of the building footprint. By modifying the footprint the city planner can change the ground plans of all floors of the building. An arbitrary wall can be selected and put into the focus of the upper-middle *lateral view*, i.e., the virtual camera is directed to that wall. The lower-middle *orthogonal wall view* focusses on the current wall, which has been selected in one of the aforementioned views. Using that view the city planner can add resp. remove or manipulate windows or doors. The upper-right *hierarchical structure view* allows a visualization of the storys in a hierarchical representation ordered by storys. The lower-right *property view* shows further information about particular parts of the building such as windows or walls. After finishing a virtual building, it can be stored and imported into a virtual 3D city. Moreover, a once generated virtual building can be stored in a virtual building library.

Besides the standard menu-based interaction concepts, such as creation, arrangement or deletion of virtual buildings, different navigation and traveling metaphors are supported. These navigation techniques include *flying*, *gliding*, *walking* metaphors and an *ufo-viewing* metaphor, i.e., an exploration with orthogonal top-view onto the city model. The proposed rear projection system, in particular the large projection screen, is beneficial for the walking metaphor via which users can explore the virtual city like a pedestrian with a large field of view. Moreover, when exploring a city model, arbitrary 3D locations can be stored as *spatial visual bookmarks* (see Section 6.4.2) to be accessed later on, for example to generate smooth camera motions along a path that can be defined by streets ([RSH05a]).

6.4 Focus and Context Visualization for Seismic Volume Datasets

Volume visualization has become a powerful tool for modern *seismic interpretation* to determine stratigraphic and structural features hidden in the dataset. Most current 3D seismic visualization techniques employ the same rendering approaches used in traditional medical imaging. This section presents extensions of the IVP metaphor proposed in Chapter 4 and its application to VR-based exploration of seismic datasets. The interpretation of seismic dataset is supported by focus and context visualization techniques in combination with intuitive and efficient VR-based interaction metaphors ([RSH06]).

The demand for seismic data interpretation is rising due to decreasing availability of fossil fuels such as oil and gas. Hence efficient techniques to locate the remaining reservoirs are needed. Drilling for these resources is a very cost intensive process, e.g., according to the *Joint Association Survey on Drilling Costs* drilling one well can cost up to several million US dollars. The oil and gas industry tries to reduce drilling costs by using extensive computer-aided analysis and exploration to improve predictions about the location of subsurface reservoirs. Also more precise predictions of natural disasters such as earthquakes can be obtained by computer-aided subsurface analysis. The most common approach to perform subsurface analysis is interactive 3D exploration of subsurface structures extracted from the seismic datasets.

The seismic data acquisition process is based on *seismic reflection phenomena*. Detonating explosives emit sound waves which are propagated through the subsurface. At interfaces between geological layers, part of the energy is refracted, another part is reflected and measured along time at surface receivers ([SMG03]). This process is repeated until the survey area is covered, and all the data is grouped and processed. The reflections of the sound waves are measured with special receivers located on the surface. Thus geological layers that reflect the sound waves can be identified based on the time elapsed before a signal is received as well as the signals attenuation. The resulting information is encoded in a 3D volume composed of discrete samples, each representing the amplitude $I(x, y, t)$ at the time t at a position (x, y) reflected beneath the surface. In some seismic processing methods the time axis is converted into depth. The value at each sample is called the *seismic amplitude*. For a fixed spatial position (x_0, y_0), the one-dimensional time-dependent function $I(x_0, y_0, t)$ is called a *seismic trace* ([SMG03, CGH86]).

This usually very large 3D volume is often stored in the *SEG-Y data exchange format* ([Com02]), and the acquired amplitudes can be visualized using *direct volume rendering (DVR)* in combination with different *transfer functions* to represent the geological structure. DVR deals with volume elements called *voxels*, the 3D analogon of the two-dimensional pixel. A variety of DVR methods exist, which all are based on the idea that voxels are assigned a color and a transparency mask with respect to certain transfer functions to determine its visual appearance ([KKH02]). Therefore even obscured voxels may contribute to the final image, and DVR allows to display an entire 3D dataset including its internals. Figure 6.10 shows a simple seismic dataset consisting of $256 \times 256 \times 256$ voxels. A grayscale transfer function is used to visualize the different geological layers.

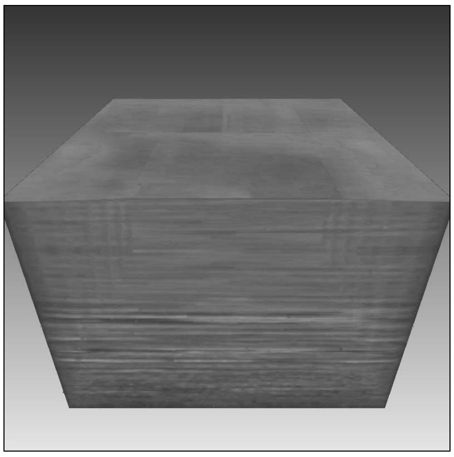


Figure 6.10: Example seismic 256^3 voxel dataset consisting of several geological layers.

Most current 3D GISs that have been proposed ([CZ05, WD04, KLR+95]) adapt the concepts of polygonal representations for visualizing geographic information, but do not support volumetric seismic datasets. For this reason highly specialized applications have been developed to explore seismic datasets by volumetric methods ([Par, 3D04]).

The applications benefit from VR systems since immersive visualization and interaction technologies allow multidisciplinary teams to explore subsurface data in a more intuitive and efficient manner and help to increase the value of such enormous amounts of volumetric datasets. As described in Chapter 2 and Chapter 3 stereoscopic projection techniques improve spatial perception and thus aid visual comprehension. Furthermore by using appropriate tracking technologies, the user can perform intuitive 3D interactions. A wide range of visualization systems and input devices, which were originally developed for applications outside of the geo-scientific industry, have been adapted to support users when exploring seismic datasets. Among others, these systems include immersive projection-based VR system environments such as the CAVE, curved and flat-wall displays, desktop or workbench-type systems as well as head-mounted devices. Especially the semi-immersive responsive workbench ([SSU99, KBF$^+$95]) has proven its potential as table-top metaphor for the exploration of geo-spatial data ([FBZ$^+$99]). The possibility to mount the workbench in a horizontal position allows visualizing geo-spatial data in an intuitive way.

Although the visualization within immersive VR systems enables an advanced exploration of seismic datasets and improves the information retrieval, interaction with the data within current systems is usually limited to mouse, keyboard and/or joystick-type devices and often requires an experienced navigation expert to pilot other viewers through the visualization. To benefit from the capabilities provided by VR systems, application developers have to consider VR-based design issues, e.g., the interactions that can be performed have to be adapted to VR-based interaction devices (see Chapter 4). Since semi-immersive and immersive VR systems allow to explore and process a huge amount of data, recent research approaches review innovative interaction technologies to enable an intuitive exploration and manipulation of geo-spatial data ([MEH$^+$99]).

For an efficient exploration of seismic datasets interactive frame rates within the application are needed in order to give immediate visual feedback when the user has performed an interaction. However, the visualization of seismic volume datasets poses three major challenges which all have a negative impact on the frame rate:

1. seismic datasets are usually very large,
2. current VR-based displays, e.g., projection screens, provide high screen resolutions, and
3. stereographic rendering used in VR systems requires to render the scene twice.

Hence it is important to use state-of-the-art volume rendering techniques, which exploit the features provided by current graphics hardware to achieve interactive frame rates.

By using the focus and context visualization metaphors presented in the next subsections the user can emphasize certain *regions of interest (RoIs)* inside the dataset interactively. RoIs can be specified by *lens volumes*, which have an arbitrary convex 3D shape. Inside the RoI a different visual representation is used for visualization and thus aids comprehension of the dataset without loosing the contextual information outside the region of interest. The user can switch between different lens types as well as visual representations inside the lens. Furthermore, it is possible to set spatial visual bookmarks at certain points of interest, which can be accessed later on using a guided VR-based exploration metaphor. For visualizing the datasets as well as the different lens types hardware-accelerated concepts ([RGW$^+$03, KW03]) have been adapted in order to enable

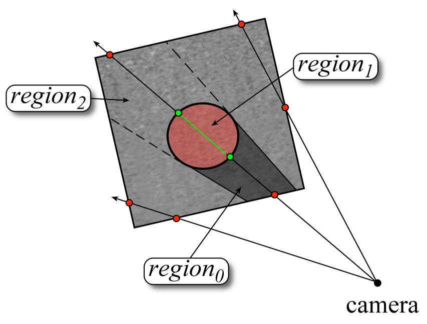


Figure 6.11: Scheme showing partition of a cast ray and additional entry and exit points at the lens volume.

interactive frame rates while rendering the scene stereoscopically and with a high resolution.

6.4.1 Visualization of Volumetric Subsurface Data

Visualization techniques for representing volumetric phenomena have advanced in the past years. While most of the proposed techniques have been developed for medical applications, Ma and Rokne ([MR04]) have presented a detailed overview of visualization techniques used to visualize seismic volume datasets. Furthermore, various software applications, e.g., *Fledermaus*[1], *VoxelGeo*[2], and *GeoProbe*[3], which use seismic data to compile and process subsurface visualizations, are available. With these applications, geo-scientists are able to visualize and interpret seismic data on commodity hardware. However, even with these highly evolved applications and the incorporated visualization techniques the information contained in volumetric data can be overwhelming during exploration. In this section an extension to the *GPU-based ray-casting technique* ([KW03, RGW$^+$03]) to support different visual representations of a RoI within a volume dataset is briefly discussed. For more detailed information regarding the algorithm the reader may consider ([RSH05c]).

Extended GPU-based Ray-Casting

GPU-based ray-casting supports very efficient volume rendering on commodity graphics hardware by casting rays through the volume dataset represented by a 3D texture ([KW03]). To apply a different visual appearance inside the lens volume, the voxels contributing to a pixel in image space need to be distinguished whether they are inside or outside the lens volume. The voxels' locations are determined before rendering the dataset, since it results in better performance because less per-fragment operations are needed during rendering compared to an approach where the regions are distinguished during rendering.

A ray cast through a volume dataset, which is intersected by a convex lens volume, is split into three different sections, i.e., one section in front of the lens, one inside the lens and one

[1]http://www.ivs3d.com/products/fledermaus/
[2]http://www.paradigmgeo.com/products/voxelgeo.php
[3]http://www.magic-earth.com/geo.asp

behind the lens (see Figure 6.11). Therefore, three view-dependent regions are distinguished:

- $region_0$: voxels in front of the lens volume,
- $region_1$: voxels inside the lens volume, and
- $region_2$: voxels behind and next to the lens volume.

The algorithm renders these view-dependent regions in three subsequent rendering passes starting with $region_0$ (see Chapter 3). Since the number of samples used for each ray depends on its length, the number of per-fragment operations required for the three sections together is equal to the number of per-fragment operations when a single ray with equal length is processed. Because the view-dependent regions of the volume dataset are determined before accessing the dataset itself, rendering can be performed very fast, since only the usually simple shaped lens geometry and the bounding box of the volume dataset are considered during this computation.

To determine the three different regions, intersection points of each ray with the lens surface are computed in volume texture coordinates ([KW03]). For a convex lens volume at most two additional intersection points per ray are required, which can be stored in two RGB textures. Since the lens volume can be defined by an arbitrary convex shape, more complex surfaces have to be handled in comparison to GPU-based ray-casting, where the entry and exit points are determined by rendering the front resp. back faces of the volume's bounding box.

As described in [RSH05b] the extended algorithm uses image-based CSG rendering techniques and similar image-based rendering techniques to determine these entry and exit points. In contrast to regular CSG rendering techniques, color channels are exploited to encode the volume texture coordinates needed during volume traversal. Because the view-dependent regions of the volume dataset are determined before accessing the dataset itself, rendering can be performed very fast, since only the usually simple shaped lens geometry and the bounding box of the volume dataset are considered during this computation.

Fröhlich et al. ([FBZ+99]) have shown how the exploration of seismic datasets can benefit from the usage of volumetric lenses. However, their approach supports only lenses having cuboid geometries. Because the surfaces of a cube are given by orthogonal planes, the visualization is similar to the common slicing approach where multiple clipping planes are used to reveal inside structures ([CGH86]). Therefore, the technique introduced in [FBZ+99] can be considered as a generalization of this slicing approach. In the next subsections concrete lenses are discussed, which are useful for 3D exploration of seismic datasets. Section 6.4.2 discusses strategies how to use these concepts in VR-based system setups.

Occlusion Lens

One problem occurring in the visualization of information associated with volumetric data is that usually no insight view can be provided. Especially when navigating through a dense volume dataset the view of the camera will always be occluded. To avoid this problem an *occlusion lens* can be used that renders those parts of the volume dataset transparently that occlude the region of interest (see Figure 6.12 (a)). Therefore when rendering $region_0$ the degree of transparency is

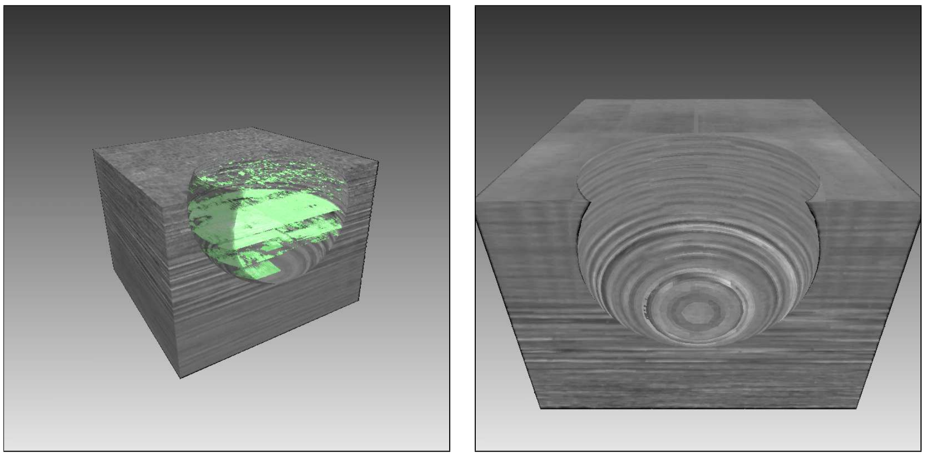

(a) Occlusion lens (b) Clipping lens

Figure 6.12: (a) occlusion and (b) clipping lenses supporting the interpretation of volumetric seismic datasets.

set proportional to the amount of volume data occluding the region of interest, i.e., the number of voxels encountered by the ray. This effect does not result in any performance penalty compared to DVR, because during ray traversal instead of setting the voxel's alpha value based on the corresponding scalar value simply the number of samples already processed is used to determine the alpha value.

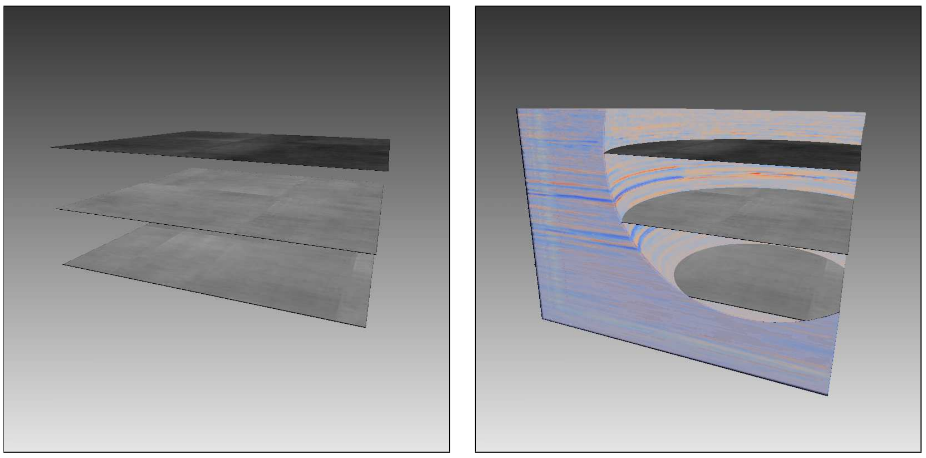
Another lens type, which is also classified as occlusion lens is the *clipping lens*, since it assists the user in exploring the information on the surface of the lens volume, which would be occluded otherwise. Such a lens is shown in Figure 6.12 (b) where a spherical clipping lens is used to reveal information hidden inside the dataset.

Slicing Lens

As shown in Figure 6.13 (a) when applying slice rendering to the entire dataset it is difficult to comprehend spatial relationships, the structure of the dataset and thus the location of the slices because parts of the dataset are invisible to the user. A *slicing lens* uses clipping-based rendering of slices as visual representation (see Figure 6.13 (b)). The spatial relationships can be detected more easily because contextual information is maintained outside the lens. Furthermore, the cross-section displayed on the surface of the lens gives an important cue to localize the visualized slices in relation to the rest of the dataset. When applying slice rendering surfaces sliced through the volume dataset are displayed. The number of slices as well as the distances between the slices can be chosen interactively and the visualization provides immediate visual feedback.

Object Emphasizing Lens

Interactive exploration of seismic volume datasets is improved by *emphasizing lenses* which emphasize RoIs by applying image-based silhouette highlighting in combination with a different



(a) Sliced seismic dataset **(b)** Slicing lens

Figure 6.13: Slicing lenses applied to a seismic dataset.

transfer function. Although in most volume visualization applications the same transfer function is used for the entire dataset, it may be desirable to use different transfer functions in different parts of the dataset ([MHB$^+$01]). Thus it is possible to reveal structures within the data without changing the context, since in the parts outside the lens volume the original transfer function is retained.

In Figure 6.14 (a) and 6.14 (b) two examples are shown where in order to highlight certain objects the parts of the dataset lying outside the lens are rendered with a different threshold value as well as a different transfer function in comparison to the parts inside the lens. In the shown examples a 1D transfer function is used inside and outside the focus region ([KKH02]). In Figure 6.14 (b) the edges between the cuboid lens shape and the surface of the dataset are enhanced. The color used for edge-enhancement, the threshold value as well as the transfer function used inside the lens volume can be exchanged interactively. Hence, geo-scientists can either start with a preset threshold and explore the highlighted structures, or they can alter the threshold interactively to find an adequate value for identifying certain structures. By combining the edge-enhancement technique with altered transfer functions objects of interest and their structure can be identified more easily (see Figure 6.14 (c)). It can be seen in Figure 6.14 (d) that using the technique for the entire dataset results in more difficult spatial comprehension since contextual information gets lost.

Level of Detail Lens

In addition to applying the visual representations as described above, the introduced algorithm allows to further enhance rendering performance. To improve performance, a different level of detail is used inside the RoI compared to the LoD outside the region. This is achieved by using a different sampling rate. Figure 6.15 shows the application of a lens with a different transfer function used for the visualization inside the lens. In Figure 6.15 (a) the same LoD is used

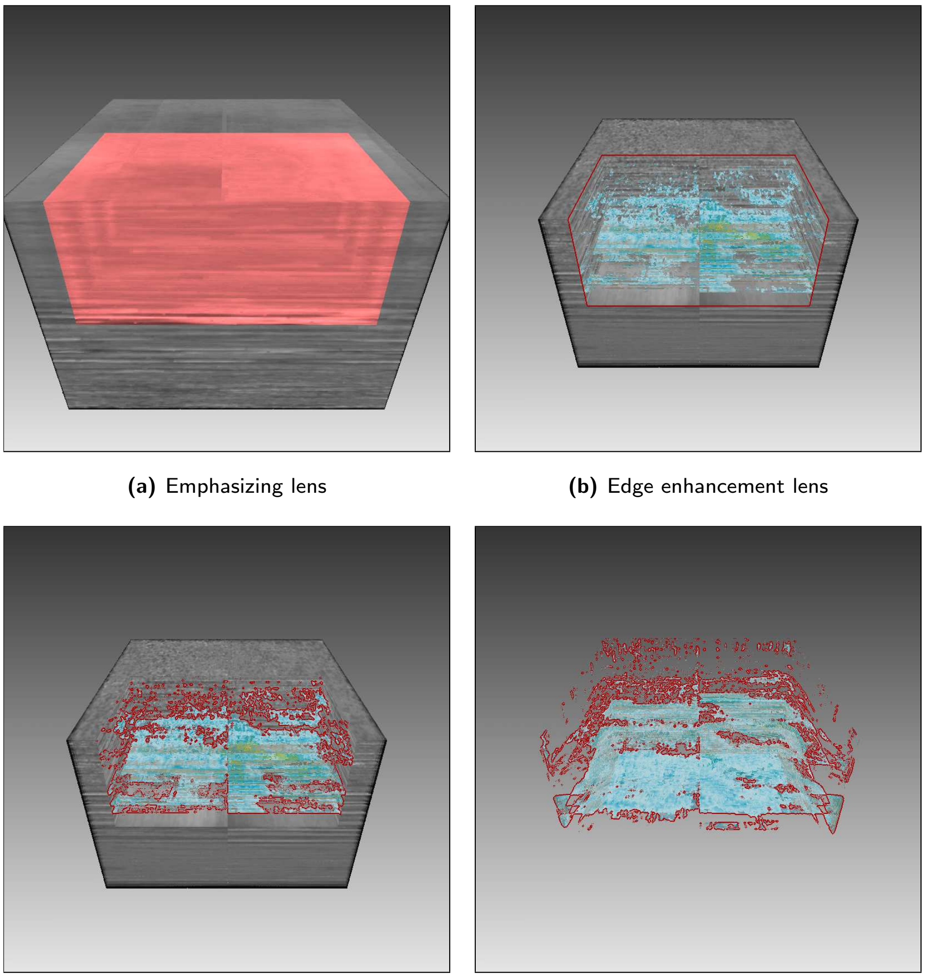
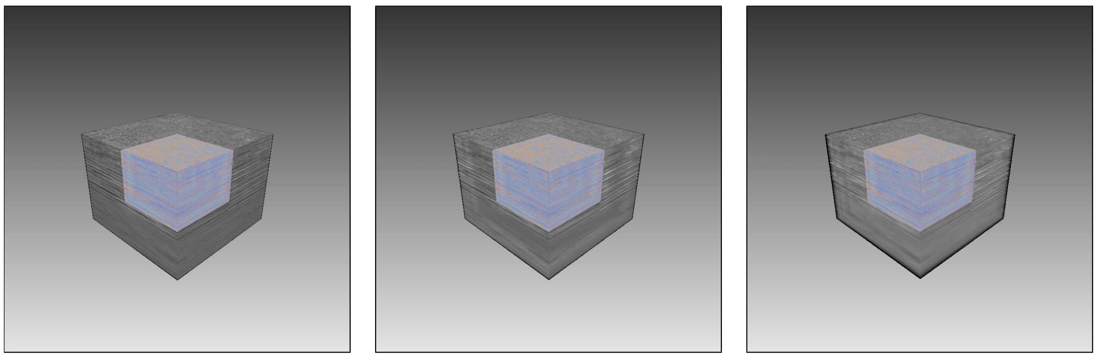

(a) Emphasizing lens **(b)** Edge enhancement lens **(c)** Enhancement lens **(d)** Entire dataset enhancement

Figure 6.14: Different emphasizing lenses using image-based edge-enhancement approaches and different transfer functions.

inside and outside the lens, i.e., 250 steps, whereas in Figure 6.15 (b) the LoD outside the lens is coarsened by using half the sampling rate compared to the visualization inside the lens. Figure 6.15 (c) illustrates the outside of the seismic dataset by using a fifth of the sampling rate for the outside of the lens compared to the visualization inside the lens. Usually, geo-scientists are interested in details within the RoI; the parts outside the region of interest are not in focus and may therefore be rendered using a lower sampling rate. Furthermore, the LoD outside the lens can be increased adaptively over time when rendering is idle.

In addition to the lenses described in this section several other lenses and even combinations of different visual representations inside the lens are possible, e.g., isosurface rendering of translucent surfaces can be combined with regular volume rendering techniques. However, the lenses presented in this section provide a good understanding of the possibilities when applying focus-



(a) 250 samples **(b)** 125 samples **(c)** 50 samples

Figure 6.15: Level of detail lens with different sampling rates outside the lens when casting rays through the dataset.

based visualization techniques to seismic volume datasets. In the next section VR-based interaction concepts based on the approaches presented in this thesis are described in order to allow an advanced exploration of seismic datasets by using the proposed lens metaphors.

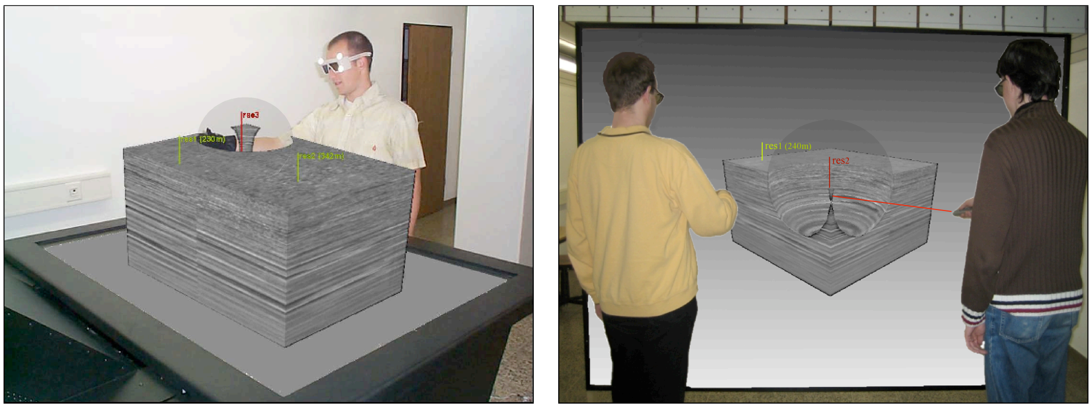
6.4.2 Interaction Strategies for VR-based Exploration

The described lens metaphor offers a comfortable way to explore seismic datasets by visually intruding into the volume. Especially in combination with a stereoscopic projection the user experiences an immersive visualization of complex structures, which improves the cognition of the spatial geodata. When using seismic exploration software interaction is often restricted to standard desktop-based interaction concepts which leads to a loss of immersion, since geoscientists have to accomplish 3D interaction using 2D devices.

Controlling the Lens

To increase the immersion, a VR-based control of the lens is presented which uses the concepts proposed in this thesis. The position and orientation of any tracked input devices are used to move an arbitrary lens accordingly. Hence the lens seems to stick at the top of the input device which ensures a very intuitive and natural mechanism for controlling the lens. However, the usage of this interaction metaphor has its limitations. In regions where the seismic data are displayed with positive stereo parallax, i.e., behind the projection screen, interaction with data positioned outside the immediate reach of the user must be guaranteed.

To enable moving the lens into the aforementioned regions, which are not accessible to the user, the C/D ratio can be changed as described in Chapter 3 to adjust the ratio determining the relationship between input device movements and the motion of the controlled virtual object, i.e., the lens volume. This adjustment is defined by the input device's position relative to the projection plane, i.e., the closer the input device is to the projection plane the higher this ratio is set. Thus it is possible to make the movements of the lens less sensitive compared to the input



(a) Responsive workbench environment

(b) Rear projection system environment

Figure 6.16: VR-based seismic exploration in (a) responsive workbench environment and (b) passive rear projection system.

device's movement or to scale the movements of the lens. In this case, a translation is applied as offset between the pose of the real input device and the virtual lens volume depending on the depth of the seismic volume dataset displayed behind the projection screen.

Spatial Visual Bookmarking

Since usually the amount of seismic data acquired for an arbitrary area is very large, relocalization of RoIs can be a difficult task. Once found regions may get lost since no VR techniques support the geo-scientist when accessing such already explored regions. Concepts for storing regions as well as positions in virtual environments have been introduced for guided navigation tasks ([Döl05a, SGLM03]). These approaches allow the user to store certain positions and to define camera paths leading through these spatial visual bookmarks placed in the VE. These concepts have been adapted to VR-based seismic exploration ([RSH06]). The following mechanism supports the user by guiding him during the exploration, i.e., the geo-scientist is assisted in relocating RoIs. Once such a region is identified, the geo-scientist can mark this region and make it accessible later on. RoIs can be stored by bookmarking them with a visual hint. In order to support an easy identification by the user textual information about the RoIs can be attached to the bookmarks (see Figure 6.16).

Individual bookmarks are markers positioned within the seismic dataset, which are easily accessible. When the lens, which is moved by the user through the dataset, emphasizes a special region of interest, an individual bookmark can be defined, e.g., by pressing a button of the input device, and the user gets a visual hint about the marker's position. This approach ensures the management of bookmarks, e.g., bookmarks can be created, deleted or repositioned.

Accessing the Bookmarks

Once the bookmarks are defined the IVP metaphor concepts discussed in Chapter 4 can be used to select a bookmark to position the lens at the appropriate location. When attempting to access a spatial visual bookmark geo-scientists benefit from the usage of the IVP metaphor since it provides a very efficient mechanism to aim at small as well as distant objects, such as spatial visual bookmarks. The user roughly points the input device to a visual bookmark, and the IVP metaphor determines the bookmark closest to the ray. Following the chosen lens is moved to this active bookmark. When using the IVP metaphor distant objects are easy to select and access to bookmarks deep inside the seismic dataset is ensured. Moreover, multimodal feedback can be given when the lens snaps to another spatial visual bookmark similar to the concepts described in Chapter 4.

In Figure 6.16 (left) a user exploits the explained interaction concepts in a responsive workbench environment. The seismic dataset is projected onto the projection screen of the workbench, and the user defines and accesses several individual bookmarks within the dataset. Figure 6.16 (right) shows two user in the passive rear projection system exploring the same seismic dataset.

Chapter 7

Conclusions

Although VR technologies provide great potential, human-computer interaction in VR-based environments makes high demands on the ability of the user to interact in a three-dimensional virtual space. Even generic interaction tasks are more difficult to accomplish in VR systems in comparison to desktop-based environments. This thesis has discussed and evaluated novel multimodal interaction metaphors for improving generic interaction tasks in virtual environments.

The generic VR software system VR^{2}S developed within the scope of this thesis eases the rapid prototyping, implementation and evaluation of intuitive interaction concepts. The system has proved its advantages in several interactive VR- and desktop-based applications. New Intuitive interaction metaphors which support the user during object selection, manipulation and exploration in VR have been developed and integrated into VR^{2}S. Case and usability studies have revealed the benefits of these metaphors for VR-based environments. In particular, the IVP metaphor concepts improve object interaction and have shown great potential to increase the acceptance of VR technologies.

This thesis has discussed the importance of multimodal interaction concepts and their integration into VR software systems demonstrated with VR^{2}S. A significant performance gain such that users could accomplish certain tasks with higher accuracy or in less time when using multimodal interaction concepts could not be observed. However, user surveys have indicated that users prefer the support via multimodal input and output. Moreover the usage of multimodal feedback has been evaluated as very helpful.

Although the proposed concepts contribute to the advancement of generic interaction tasks, there are still challenging issues concerning the interaction processes in VR-based environments. In the future further interaction concepts will be integrated into VR^{2}S. These strategies will incorporate constraint-based interaction approaches and focus on domain-specific interactions, e.g., medical exploration tasks. In addition, further multimodal input devices that are optimized for the demands of domain-specific users will be developed and evaluated in different application domains.

Future research may aim to develop virtual reality hardware that will open up new vistas for HCI. In order to increase the acceptance of these sophisticated technologies the user will be less bothered with special devices. For instance, tracking technologies should enable users to interact without the requirement for equipping them with markers or input devices. Tracking systems should interpret arbitrary gestures of users in order to allow them to accomplish desired tasks. Output devices that address all human sensorial channels will be pervasive, and thus the user will not be restricted to specific VR system environments. For example, enhancements of autostereoscopic display technologies may enable several users to view high-quality view-dependent stereoscopic images without the need of wearing glasses or HMDs. Sophisticated computer systems supporting multimodal interaction may become ubiquitous. The metaphors proposed within this thesis are not restricted to special VR hardware such as the described setups. For instance, pointer metaphors can also be performed by natural gestures, e.g., pointing with the index finger. Thus the concepts and strategies developed within this thesis will still be relevant if future achievements evolve completely different and more advanced technology.

Bibliography

[3D04] IVS 3D. *Fledermaus Reference Manual*, 2004.

[3DC] 3DConnexion. Products (http://www.3dconnexion.de/products/3a.php).

[AB91] G. Abowd and R. Beale. Users, Systems and Interfaces: A Unifying Framework for Interaction. In D. Diaper and N. Hammond, editors, *Proceedings of HCI '91 People and Computers VI*, pages 73–87, 1991.

[ABF$^+$94] P. Astheimer, K. Böhm, W. Felger, M. Göbel, and S. Müller. Die Virtuelle Umgebung - Eine neue Epoche in der Mensch-Maschine-Kommunikation. *Informatik-Spektrum*, 17(6), 1994.

[ABF$^+$97] M. Agrawala, A. Beers, B. Fröhlich, F. Klimetzek, and M. Bolas. The Two-User Responsive Workbench: Support for Collaboration through Individual Views of Shared Space. In *ACM Proceedings of Computer Graphics and Interactive Techniques*, pages 327 – 332, 1997.

[AKO95] Y. Adachi, T. Kumano, and K. Ogino. Intermediate Representation for Stiff Virtual Objects. In *Proceedings of IEEE VRAIS*, pages 195–203, 1995.

[AMH02] T. Akenine-Möller and E. Haines. *Real-Time Rendering.* A. K. Peters Ltd., 2002.

[BBN02] P. Barr, R. Biddle, and J. Noble. A Taxonomy of User-Interface Metaphors. Technical Report CS-TR-02-1, School of Mathematical and Computing Sciences, Victoria University, 2002.

[BC03] G. Burdea and P. Coiffet. *Virtual Reality Technology.* Wiley-IEEE Press, 2003.

[Beg92] D. Begault. An Introduction to 3-D Sound for Virtual Reality. Technical report, NASA-Ames Research Center, 1992.

[Beg94] D. Begault. *3-D Sound for Virtual Reality and Multimedia.* Morgan Kaufmann Pub, 1994.

[BFSE01] O. Bimber, B. Fröhlich, D. Schmalstieg, and M. Encarnação. The Virtual Showcase. *IEEE Computer Graphics and Applications*, 21(6):48–55, 2001.

[BH93] K. Böhm and W. Hübner. Virtual Reality: A New User Interface Paradigm for Industrial Applications. In *Proceedings of Imagina 93*, pages 69–84, 1993.

[BH95] D. Bowman and L. Hodges. User Interface Constraints for Immersive Virtual Environment Applications. Technical Report GIT-GVU-95-26, Graphics, Visualization, and Usability Center, 1995.

[BH97] D. Bowman and L. Hodges. An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments. In *ACM Symposium on Interactive 3D Graphics*, pages 35–38, 1997.

[BH99] D. Bowman and L. Hodges. Formalizing the Design, Evaluation, and Application of Interaction Techniques for Immersive Virtual Environments. *Journal of Visual Languages and Computing*, 10, 1999.

[BJ98] A. Bierbaum and C. Just. Software Tools for Virtual Reality Application Development. In *Course Notes for SIGGRAPH 98 Course 14, Applied Virtual Reality*, 1998.

[BJH$^+$01] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *IEEE Proceedings of VR2001*, pages 89–96, 2001.

[BKLP01] D. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. An Introduction to 3-D User Interface Design. *Presence: Teleoperators and Virtual Environments*, 10(1):96–108, 2001.

[BKLP04] D. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley, 2004.

[BLRS98] R. Blach, J. Landauer, A. Roesch, and A. Simon. A Flexible Prototyping Tool for 3D Real-Time User Interaction. *ACM Proceedings of Eurographics Workshop of Virtual Environments (VE'98)*, pages 195–203, 1998.

[BM86] W. Buxton and B. Myers. A Study in Two-Handed Input. In *Human Factors in Computing Systems*, pages 321–326, 1986.

[BMS00] W. Broll, E. Meier, and T. Schardt. The Virtual Round Table - A Collaborative Augmented Multi-User Environment. *ACM Proceedings of Collaborative Virtual Environments (CVE2000)*, pages 39–45, 2000.

[BN76] J. Blinn and M. Newell. Texture and Reflection in Computer Generated Images. *Communications of the ACM*, 19:542–546, 1976.

[Bou99] P. Bourke. Calculating Stereo Pairs (http://astronomy.swin.edu.au/ pbourke/stereographics/stereorender/), 1999.

[Bow99] D. Bowman. *Interaction Techniques for Common Tasks in Immersive Virtual Environments: Design, Evaluation, and Application.* PhD thesis, Georgia Institute of Technology, 1999.

[Cal05] M. Calis. Haptics. Technical report, Heriot-Watt University Internal Report, 2005.

[Car01] J. Carroll. *Human-Computer Interaction in the New Millenium.* Addison-Wesley Professional, 2001.

[CGH86] M. Curtis, A. Gerhardstein, and R. Howard. Interpretation of Large 3-D Data Volumes. In *Expanded Abstracts of the Society of Exploration Geophysicists 56th Annual Meeting*, pages 497–499, 1986.

[Cha92] H. J. Charwat. *Lexikon der Mensch-Maschine-Kommunikation.* Oldenbourg, 1992.

[CMK88] J. Carroll, R. Mack, and W. Kellogg. *Handbook of Human-Computer Interaction*, chapter Interface Metaphors and User Interface Design, pages 67–85. Elsevier Science Publishers, 1988.

[CMN83] S. Card, T. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, 1983.

[CN95] C. Cruz-Neira. *Virtual Reality based on Multiple Projection Screens: The CAVE and Its Applications to Computational Science and Engineering.* PhD thesis, University of Illinois at Chicago, 1995.

[CNSD$^+$92] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. Kenyon, and J. C. Hart. The CAVE, Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, pages 64–72, June 1992.

[Com87] Apple Computer. *Human Interface Guidelines: The Apple Desktop Metaphor.* Addison Wesley, 1987.

[Com02] SEG Technical Standards Committee. *SEG Y rev 1 Data Exchange Format*, 2002.

[CS73] L. Cooper and R. Shepard. Chronometric Studies of the Rotation of Mental Images. *Visual information processing*, pages 75–176, 1973.

[CZ05] V. Coors and A. Zipf, editors. *3D-Geoinformationssysteme - Grundlagen und Anwendungen.* Wichmann Verlag, 2005.

[Dah06] M. Dahm. *Grundlagen der Mensch-Computer-Interaktion.* Pearson Studium, 2006.

[DFAB98] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction.* Prentice Hall, 1998.

[DFS01] R. De Amicis, M. Fiorentino, and A. Stork. Parametric Interaction for CAD Application in Virtual Reality Environment. In *XII ADM INTERNATIONAL CONFERENCE on Design Tools and Methods in Industrial Engineering*, pages D3/43–D3/52, 2001.

[DH98] J. Döllner and K. Hinrichs. Interactive, Animated 3D Widgets. In *Computer Graphics International 1998*, pages 278–286, 1998.

[DH02] J. Döllner and K. Hinrichs. A Generic Rendering System. *IEEE Transaction on Visualization and Computer Graphics*, 8(2):99–118, 2002.

[dHKP05] G. de Haan, M. Koutek, and F. Post. IntenSelect: Using Dynamic Object Rating for Assisting 3D Object Selection. In E. Kjems and R. Blach, editors, *Proceedings of the 9th IPT and 11th Eurographics VE Workshop (EGVE) '05*, pages 201–209, 2005.

[Döl05a] J. Döllner. Constraints as Means of Controlling Usage of Geovirtual Environments. *Journal of Cartography and Geographic Information Science*, 32(2):69–80, 2005.

[Döl05b] J. Döllner. *VRS Documentation (http://www.vrs3d.org/)*, 2005.

[DU02] K. Dorfmüller-Ulhaas. *Optical Tracking - From User Motion to 3D Interaction.* PhD thesis, Technische Universität Wien, 2002.

[Ebe94] R. Eberts. *User Interface Design*, chapter Experimental Designs and Analysis, pages 82–125. Prentice Hall, 1994.

[EEB67] W. English, D. Engelbart, and M. Berma. Display-Selection Techniques for Text Manipulation. *IEEE Transactions on Human Factors in Electronics*, HFE-8(1):5–15, 1967.

[Fau95] N. Faust. The Virtual Reality of GIS. *Environment and Planning B: Planning and Design*, 22:257–268, 1995.

[FBZ+99] B. Fröhlich, S. Barrass, B. Zehner, J. Plate, and M. Göbel. Exploring Geo-Scientific Data in Virtual Environments. In *IEEE Visualization*, pages 169–173, 1999.

[FD82] J. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1982.

[FHH+05] B. Fröhlich, J. Hochstrate, J. Hoffmann, K. Krüger, R. Blach, M. Bues, and O. Stefani. Implementing Multi-User Stereo Displays. *13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 139–146, 2005.

[FHZ96] A. Forsberg, K. Herndorn, and R. Zeleznik. Aperture based Selection for Immersive Virtual Environments. In *ACM Symposium on User Interface Software and Technology*, pages 95–96, 1996.

[Fit54] P. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

[FK05] S. Frees and G. Kessler. Precise and Rapid Interaction through Scaled Manipulation in Immersive Virtual Environments. In *IEEE Virtual Reality 2005*, pages 99–106, 2005.

[FP01] A. Fuhrmann and W. Purgathofer. Studierstube: An Application Environment for Multi-User Games in Virtual Reality. *GI-Jahrestagung*, 2:1185–1190, 2001.

[FWC84] J. Foley, V. Wallace, and P. Chan. The Human Factors of Computer Graphics Interactive Techniques. In *Computer Graphics and Applications*. IEEE, 1984.

[GHJV93] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. *Lecture Notes in Computer Science*, 707:406–431, 1993.

[GLM97] G. Goebbels, V. Lalioti, and T. Mack. Guided Design and Evaluation of Distributed, Collaborative 3D Interaction in Projection based Virtual Environments. *ACM Siggraph Proceedings*, 15(5):745–770, 1997.

[Gol03] E. Goldstein. *Sensation and Perception*. Wadsworth, 2003.

[Gre86] N. Greene. Applications of World Projections. In *Proceedings Graphics Interface Proceedings Graphics Interface*, pages 108–114, 1986.

[Gro05] Open Inventor Architecture Group. *Open Inventor C++ Reference Manual*. Addison-Wesley, 2005.

[HBC^{+}92] T. Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei G. Perlman, G. Strong, and W. Verplank. ACM SIGCHI Curricula for Human-Computer Interaction. Technical report, ACM Order Number: 608920, 1992.

[Hei92] M. Heilig. El Cine del Futuro: The Cinema of the Future. *Presence*, 1(3):279–294, 1992.

[Hel95] J. Helman. Performance Requirements and Human Factors, 1995.

[Hie05] G. Hiebert. *OpenAL 1.1 Specification and Reference*, 2005.

[HL95] R. Holloway and A. Lastra. Virtual Environments: A Survey of the Technology, 1995.

[HM85] L. Hodges and D. McAllister. Stereo and Alternating Pair Techniques for Display of Computer Generated Images. *IEEE Computer Graphics and Applications*, 5(9):38–45, 1985.

[Hou92] S. Houde. Iterative Design of an Interface for Easy 3-D Direct Manipulation. In *Conference on Human Factors in Computing Systems (CHI'92)*, pages 135–142, 1992.

[Hug05] D. Hughes. Defining an Audio Pipeline for Mixed Reality. In *Proceedings of Human Computer Interfaces International*, 2005.

[IS94] M. Ishii and M. Sato. A 3D Space Interface Device Using Tensed Strings. *Presence*, 3(81-86), 1994.

[JFH94] R. Jacoby, M. Ferneau, and J. Humphries. Gestural Interaction in a Virtual Environment. In *Stereoscopic Displays and Virtual Reality Systems: The Engineering of Virtual Reality*, pages 355–364. SPIE, 1994.

[JL04] K. Jacobs and C. Loscos. Classification of Illumination Methods for Mixed Reality. Technical report, Eurographics 2004, 2004.

[Jon01] L. Jones. Human Factors and Haptic Interfaces. IMA Talks Haptics, Virtual Reality, and Human Computer Interaction, 2001.

[JvDFH92] J.Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1992.

[Kal93] R. Kalawsky. *The Science of Virtual Reality and Virtual Environments*. Addison-Wesley Publishing Company, 1993.

[KASK02] J. Kelso, L. Arsenault, S. Satterfield, and R. Kriz. DIVERSE: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments. In *Proceedings of Virtual Reality 2002 Conference*, pages 183–190. IEEE, 2002.

[Kau03] H. Kaufmann. Collaborative Augmented Reality in Education. Technical Report 188-2-2003-1, Technische Universität Wien, 2003.

[KB99] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In *In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR)*, 1999.

[KBF$^+$95] W. Krüger, C. Bohn, B. Fröhlich, H. Schuth, W. Strauss, and G. Wesche. The Responsive Workbench: A Virtual Work Environment. *IEEE Computer*, 28(8):42–48, 1995.

[KD02] O. Kersting and J. Döllner. Interactively Developing 3D graphics Applications in Tcl. In *USENIX Annual Technical Conference*, pages 99–118, 2002.

[KJM03] B. Kapralos, M. Jenkin, and E. Milios. Auditory Perception and Spatial 3D Auditory Systems. Technical Report CS-2003-07, York University, 2003.

[KKH02] J. Kniss, G. Kindlmann, and C. Hansen. Multi-Dimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):270–285, 2002.

[KKYK01] Y. Kitamura, T. Konishi, S. Yamamoto, and F. Kishino. Interactive Stereoscopic Display for Three or More Users. In *SIGGRAPH*, pages 231–239. ACM, 2001.

[KLR+95] D. Koller, P. Lindstrom, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Virtual GIS: A Real-time 3D Geographic Information System. In *Visualization*, pages 94–100. IEEE, 1995.

[KW03] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings of IEEE Visualization*, pages 287–292, 2003.

[Lat01] M. Latoschik. *Multimodale Interaktion in Virtueller Realität am Beispiel der Virtuellen Konstruktion*. PhD thesis, Universität Bielfeld, 2001.

[LG94] J. Liang and M. Green. JDCAD: A Highly Interactive 3D Modeling System. *Computers & Graphics*, 18(4):499–506, 1994.

[Lik32] R. Likert. A Technique for the Measurement of Attitude. *Archives of Psychology*, 140:1–55, 1932.

[Lip91] L. Lipton. *The CrystalEyes Handbook*. Stereographics Corportation, 1991.

[LJB+99] J. Leigh, A. Johnson, M. Brown, D. Sandin, and T. DeFanti. Visualization in Teleimmersive Environments. *IEEE Computer*, 32(12):66–73, 1999.

[McG79] M. McGee. Human Spatial Abilities: Psychometric Studies and Environmental, Genetic, Hormonal and Neurological Influences. *Psychological Bulletin*, 5:889–918, 1979.

[MDG+95] P. Milgram, D. Drascic, J. Grodski, A. Restogi, S. Zhai, and C. Zhou. Merging Real and Virtual Worlds. In *Proceedings of IMAGINA'95*, 1995.

[MEH+99] A. MacEachren, R. Edsall, D. Haug, R. Baxter G. Otto, R. Masters, S. Fuhrmann, and L. Qian. Virtual Environments for Geographic Visualization: Potential and Challenges. *Proceedings of the ACM Workshop on New Paradigms for Invormation Visualization and Manipulation*, pages 35–40, 1999.

[MG96] T. Mazuryk and M. Gervautz. Virtual Reality History, Applications, Technology and Future. Technical Report TR-186-2-96-06, TU Wien, 1996.

[MHB+01] D. Meyer, E. Harvey, T. Bulloch, J. Voncannon, and T. Sheffield. Use of Seismic Attributes in 3-D Geovolume Interpretation. *The Leading Edge*, 20(12):1377–1381, 2001.

[Mic05] Microsoft. *MSDN Library*, 2005.

[Mil56] G. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review*, 63(2):81 – 97, 1956.

[MIO+04] M. Matsushita, M. Iida, T. Ohguro, Y. Shirai, Y. Kakehi, and T. Naemura. Lumisight Table: Interactive View-dependent Display-Table Surrounded by Multiple Users. In *SIGGRAPH*. ACM, 2004.

[MK94] P. Milgram and F. Kishino. A Taxonomy of Mixed Reality Visual Displays. In *IEICE Transactions on Information and Systems, Special issue on Networked Reality*, 1994.

[MP93] S. Musil and G. Pigel. Virgets: Elements for Building 3D User Interfaces. In *VR 1993*. IEEE, 1993.

[MR04] C. Ma and J. Rokne. *3D Seismic Volume Visualization*. Kluwer Academic, 2004.

[MVS$^+$02] J. Mortensen, V. Vinayagamoorthy, M. Slater, A. Steed, B. Lok, and M. C. Whitton. Collaboration in Tele-Immersive Environments. *ACM Proceedings of Workshop on Virtual Environments*, pages 93–101, 2002.

[Nad00] D. Nadeau. Volume Scene Graphs. In *Volume Visualization*, pages 49–56, 2000.

[Nie86] J. Nielsen. A Virtual Protocol Model for Computer-Human Interaction. *International Journal of Man-Machine Studies archive*, 24(3):301–312, 1986.

[NKH98] T. Naemura, M. Kaneko, and H. Harashima. Multi-User Immersive Stereo in Projection-based Virtual Reality System. *Conference Proceedings of The Virtual Reality Society of Japan*, 3(8):57–60, 1998.

[NNMH02] T. Naemura, T. Nitta, A. Mimura, and H. Harashima. Virtual Shadows in Mixed Reality Environment Using Flashlight-Like Devices. *Transaction of Virtual Reality Society of Japan*, 7(2):227–237, 2002.

[Nor98] D. Norman. *The Design of Every-Day Things*. PhD thesis, MIT, 1998.

[Nor99] D. Norman. Affordance, Conventions, and Design. *Interactions*, 6(3):38–42, 1999.

[NS97] C. North and B. Shneiderman. A Taxonomy of Multiple Window Coordinations. Technical Report CS-TR-3854, University of Maryland Computer, 1997.

[OF03] A. Olwal and S. Feiner. The Flexible Pointer: An Interaction Technique for Selection in Augmented and Virtual Reality. In *ACM Symposium on User Interface Software and Technology (Conference Supplement)*, pages 81–82, 2003.

[ON01] D. Olsen and S. Nielsen. Laser Pointer Interaction. In *Proceedings of CHI*, pages 17–22, 2001.

[Par] ParadigmGeo. VoxelGeo: Volume-based Seismic Interpretation (http://www.paradigmgeo.com/objects/pdf/voxelgeo.pdf).

[PBC$^+$95] R. Pausch, T. Burnette, A. C. Capehar, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J. White. A Brief Architectural Overview of Alice, a Rapid Prototyping System for Virtual Reality. In *IEEE Computer Graphics and Applications*, pages 195–203, 1995.

[PBWI96] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR. In *ACM Symposium on User Interface Software and Technology*, pages 79–80, 1996.

[PFC$^+$97] J. Pierce, A. Forsberg, M. Conway, S. Hong, R. Zeleznik, and M. Mine. Image Plane Interaction Techniques in 3D Immersive Environments. In *ACM Symposium on Interactive 3D Graphics*, pages 39–44, 1997.

[PHH$^+$03] J. Piesk, R. Heeg, R. Hönscheid, N. Krämer, and G. Bente. Echtzeit-Visualisierungs-Editor für Deskriptive Verhaltensprotokolle. In *Mensch & Computer*, 2003.

[PNMI05] J. Parker, M. Nunes, R. Mandryk, and K. Inkpen. TractorBeam Selection Aids: Improving Target Acquisition for Pointing Input on Tabletop Displays. In *Proceedings of INTERACT 2005*, pages 80–93, 2005.

[Pop01] V. Popescu. *Design and Performance Analysis of a Virtual Reality-based Telerehabilitation System.* PhD thesis, Rutgers University, 2001.

[PWBI98] I. Poupyrev, S. Weghorst, M. Billinghurst, and T. Ichikawa. Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques. *Computer Graphics Forum*, 17(3):41–52, 1998.

[Rai99] R. Raisamo. *Multimodal Human-Computer Interaction: A Constructive and Empirical Study.* PhD thesis, University of Tampere, 1999.

[RBML04] I. Rebelo, R. Barcia, E. Merino, and R. Luz. Evaluation of VR Systems: More Usable Interactions. In *VRIC*, pages 241–249. IEEE, 2004.

[RGW$^+$03] S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Straßer. Smart Hardware-Accelerated Volume Rendering. In *Proceedings of the Symposium on Visualization*, pages 231–238. IEEE Press, 2003.

[RHS05] T. Ropinski, K. Hinrichs, and F. Steinicke:. A Solution for the Focus and Context Problem in Virtual Geo-Environments. In *Proceedings of the 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS05)*, 2005.

[Rol05] S. Rolfes. Integration Indirekter und Multimodaler Interaktionskonzepte in das Objekt-orientierte Grafiksystem VRS. Master's thesis, WWU Münster, 2005.

[RSH05a] T. Ropinski, F. Steinicke, and K. Hinrichs. A Constrained Road-based VR Navigation Technique for Travelling in 3D City Models. In *Proceedings of the 15th International Conference on Artificial Reality and Telexistence (ICAT05)*, pages 228–235, 2005.

[RSH05b] T. Ropinski, F. Steinicke, and K. Hinrichs. Interactive Importance-Driven Visualization Techniques for Medical Volume Data. In *Proceedings of the 10th International Fall Workshop on Vision, Modeling, and Visualization (VMV05)*, pages 273–280, 2005.

[RSH05c] T. Ropinski, F. Steinicke, and K. Hinrichs. Tentative Results in Focus-based Medical Volume Visualization. In *Proceedings of 5th International Smart Graphics Symposium, Lecture Notes in Computer Science 3638*, pages 218–221, 2005.

[RSH06] T. Ropinski, F. Steinicke, and K. Hinrichs. Visual Exploration of Seismic Volume Datasets. *Journal of the 14th International Conference in Central Europe on Computer Graphics Visualization and Computer Vision (WSCG06)*, pages 73–80, 2006.

[RWH04] T. Ropinski, S. Wachenfeld, and K. Hinrichs. Virtual Reflections for Augmented Reality Environments. In *Proceedings of the 14th International Conference on Artificial Reality and Telexistence (ICAT04)*, pages 311–318, 2004.

[SCB04] K. Salisbury, F. Conti, and F. Barbagli. Haptic Rendering: Introductory Concepts. *IEEE Computer Graphics and Applications*, 24(2):24–32, 2004.

[Sch01] D. Schmalstieg. *Collaborative Augmented Reality.* PhD thesis, Technische Universität Wien, Habilitation thesis, 2001.

[SFH$^+$02] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavari, M. Encarnação, M. Gervautz, and W. Purgathofer. The Studierstube Augmented Reality Project. In *PRESENCE - Teleoperators and Virtual Environments 11(1)*, pages 32–45, 2002.

[SG97] Z. Szalavári and M. Gervautz. Using the Personal Interaction Panel for 3D Interaction. In *Proceedings of the Conference on Latest Results in Information Technology*, page 36, 1997.

[SGI04] SGI. *A Technical Overview of OpenGL Performer 3.1.* SGI White Paper, 2004.

[SGLM03] B. Salomon, M. Garber, M. Lin, and D. Manocha. Interactive Navigation in Complex Environments Using Path Planning. In *Symposium on Interactive 3D Graphics*, pages 41–50, 2003.

[SGLS93] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled Simulation in Virtual Reality with the MR Toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, 1993.

[Shi92] K. Shimoga. Finger Force and Touch Feedback Issues in Dexterous Telemanipulation: A Survey. In *Proceedings of the NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration*, pages 159–178, 1992.

[Shn97a] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1997.

[Shn97b] B. Shneiderman. Direct Manipulation for Comprehensible, Predictable and Controllable User Interfaces. In *Intelligent User Interfaces*, pages 33–39, 1997.

[Shr04] D. Shreiner. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.4, 4/E.* Addison Wesley, 2004.

[SHR05] F. Steinicke, K. Hinrichs, and T. Ropinski. Virtual Reflections and Virtual Shadows in Mixed Reality Environments. In *10th International Conference on Human-Computer Interaction (INTERACT2005)*, pages 1018–1021, 2005.

[SHR06] F. Steinicke, K. Hinrichs, and T. Ropinski. A Hybrid Decision Support System for 3D City Planning. In *ISPRS Commission II Symposium on Spatio-temporal Data Handling and Information*, page (in print), 2006.

[Sil79] D. Silbernagel. *Taschenatlas der Physiologie.* Thieme, 1979.

[SMG03] R. Silva, M. Machado, and M. Gattass. 3D Seismic Volume Rendering. In *Proceedings of the Eight International Congress of the Brazilian Geophysical Society (CSBGS)*, 2003.

[Smi04] R. Smith. *Open Dynamics Engine v0.039 User Guide*, 2004.

[SNC$^+$95] L. Schomaker, J. Nijtmans, A. Camurri, F. Lavagetto, P. Morasso, C. Benoit, T. GuiardMarigny, B. Le Goff, J. Robert-Ribes, A. Adjoudani, I. Defee, S. Münch, K. Hartung, and J. Blauert. A Taxonomy of Multimodal Interaction in the Human Information Processing System. Technical Report Basic Research Project 8579, Miami ESPRIT III, 1995.

[Sou92] D. Southard. Transformations for Stereoscopic Visual Simulation. *Computer & Graphics*, 16(4):401–410, 1992.

[SRD00] H. Sowizral, K. Rushforth, and M. Deering. *The Java 3D API Specification.* Addison-Wesley, 2000.

[SRH04] F. Steinicke, T. Ropinski, and K. Hinrichs. Object Selection in Virtual Environments with an Improved Virtual Pointer Metaphor. In *Proceedings of International Conference on Computer Vision and Graphics (ICCVG)*, pages 320–326, 2004.

[SRH05a] F. Steinicke, T. Ropinski, and K. Hinrichs. A Generic Virtual Reality Software System's Architecture and Application. In *Proceedings of the 15th International Conference on Artificial Reality and Telexistence (ICAT05)*, pages 220–227, 2005.

[SRH05b] F. Steinicke, T. Ropinski, and K. Hinrichs. Co-located Interaction Concepts for Large Screen Displays. Technical Report 06/05 - I, WWU Münster, 2005.

[SRH05c] F. Steinicke, T. Ropinski, and K. Hinrichs. Multimodal Interaction Metaphors for Manipulation of Distant Objects in Immersive Virtual Environments. In *13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 45–48, 2005.

[SRH05d] F. Steinicke, T. Ropinski, and K. Hinrichs. VR and Laser-based Interaction in Virtual Environments using a Dual-Purpose Interaction Metaphor. In *VR2005 Workshop Proceedings on New Directions in 3D User Interfaces*, pages 61–64. IEEE, 2005.

[SRH06] F. Steinicke, T. Ropinski, and K. Hinrichs. Collaborative Interation Concepts for City Planning Tasks in Projection-based Virtual Reality Systems. In *8th International Conference on Virtual Reality*, pages 113–122. IEEE, 2006.

[SRHB06] F. Steinicke, T. Ropinski, K. Hinrichs, and G. Bruder. A Multiple View System for Modeling Building Entities. In *Proceedings of 4th International Conference on Coordinated & Multiple Views in Exploratory Visualization*, page (in print). IEEE, 2006.

[SRHM06] F. Steinicke, T. Ropinski, K. Hinrichs, and J. Mensmann. Urban City Planning in Semi-immersive Virtual Reality. In *Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP2006)*, pages 192–199. INSTICC Press, 2006.

[SS05] A. Simon and S. Scholz. Multi-Viewpoint Images for Multiple-User Interaction. In *IEEE Proceedings of VR2005*, pages 107–113, 2005.

[SSC01] M. Slater, A. Steed, and Y. Chrysanthou. *Computer Graphics and Virtual Environments: From Realism to Real-Time.* Addison-Wesley, 2001.

[SSS01] G. Smith, T. Salzman, and W. Stürzlinger. 3D Scene Manipulation with Constraints. In B. Fisher, K. Dawson-Howe, and C. O'Sullivan, editors, *Virtual and Augmented Architecture*, pages 35–46, 2001.

[SSU99] B. Schmidt, U. Streit, and C. Uhlenküken. Zur Einsetzbarkeit von Workbench-Umgebungen für Geowissenschaftliche und Planerische Fragestellungen. In *Symposium für Angewandte Geographische Informationsverarbeitung*, pages 482–489, 1999.

[Sut63] I. Sutherland. *Sketchpad: A Man-Machine Graphical Communications System.* PhD thesis, M.I.T., 1963.

[Sut65] I. Sutherland. The Ultimate Display. In *Proceedings of IFIP Congress 2*, pages 506–509, 1965.

[Sut68] I. Sutherland. A Head-Mounted Three-Dimensional Display. In *Proceeding of the Fall Joint Computer Conference*, volume 33, pages 757–764. AFIPS, 1968.

[Ter05] L. Terrenghi. Design of Affordances for Direct Manipulation of Digital Information in Ubiquitous Computing Scenarios. In *Proceedings of 5th International Smart Graphics Symposium, Lecture Notes in Computer Science 3638*, pages 198–205, 2005.

[TH92] T. Takala and J. Hahn. Sound Rendering. In *19th International Conference on Computer Graphics and Interactive Techniques*, pages 211–220, 1992.

[Tra99] H. Tramberend. AVANGO: A distributed virtual reality framework. In *Proceedings of the IEEE Virtual Reality '99*, 1999.

[Vin95] J. Vince. *Virtual Reality Systems.* Addision-Wesley, 1995.

[WD04] M. Worboys and M. Duckham. *GIS: A Computing Perspective.* CRC Press, 2004.

[Wil78] L. Williams. Casting Curved Shadows on Curved Surfaces. In *Proceedings of SIGGRAPH 78*, volume 12, pages 270–274, 1978.

[Wis01] M. Wissen. Implementation of a Laser-based Interaction Technique for Projection Screens. *ERCIM News*, 46:31–32, 2001.

[YNTT03] Y. Yanagida, H. Noma, N. Tetsutani, and A. Tomono. Computers Everywhere: An Unencumbering, Localized Olfactory Display. In *CHI - Interactive posters*, 2003.

[Zel92] D. Zeltzer. Autonomy, Interaction, and Presence. *Presence*, 1(1):127–132, 1992.

[ZTW$^+$01] P. Zahorik, C. Tam, K. Wang, P. Bangayan, and V. Sundareswaran. Localization Accuracy in 3D Sound Displays: The Role of Visual-Feedback Training, 2001.

Lebenslauf

Persönliche Daten

Name	:	Frank Steinicke
Geburt	:	16. Mai 1977 in Rheine
Familienstand	:	ledig
Eltern	:	Johann Otto Steinicke
	:	Alwine Johanne Frida Steinicke, geb. Schulte Siering

Schulausbildung

07/1983 - 06/1987	:	Grundschule auf dem Süsteresch, Schüttorf
07/1987 - 06/1996	:	Städtisches Gymnasium, Ochtrup
21. Mai 1996	:	Hochschulreife (Abitur)

Zivildienst

09/1996 - 10/1997	:	Gemeinschaftsgrundschule Paul-Gerhard, Emsdetten

Studium

10/1997-12/2002	:	Studium Diplom-Mathematik mit Nebenfach Informatik, Westfälische Wilhelms-Unversität, Münster
2. Dezember 2002	:	Diplom in Mathematik, Westfälische Wilhelms-Unversität, Münster
seit 01/2003	:	Beginn der Dissertation bei Prof. Dr. Klaus H. Hinrichs, Institut für Informatik, Westfälische Wilhelms-Unversität, Münster

Berufliche Tätigkeiten

07/1994 - 06/1999	:	Handballspieler, TV Emsdetten GmbH & Co KG, Emsdetten
seit 07/1999	:	Handballspieler, TuS Spenge e.V., Spenge
11/2004 - 03/2005	:	wissenschaftliche Hilfskraft, Arbeitsgruppe Grafische und Geometrische Datenverarbeitung, Institut für Informatik, Westfälische Wilhelms-Unversität, Münster
seit 04/2005	:	wissenschaftlicher Mitarbeiter, Arbeitsgruppe Grafische und Geometrische Datenverarbeitung, Institut für Informatik, Westfälische Wilhelms-Unversität, Münster