Scientific Visualization in Virtual Reality:

Interaction Techniques and Application Development

About the cover image
Metaphorical illustration of scientific visualization in Virtual Reality. A display can be considered as a window into a virtual world, see Chapters 1 and 2.

About the front cover
The Virtual Workbench is in use for interactive exploration and visualization of a cumulus clouds dataset, see Section 6.3. The VRX toolkit (Section 5.2) is used for this purpose.
Central image: A user at the Workbench is studying the vertical air velocity in the interior of a cloud using a direct slicing tool, attached to the Plexipad, see Section 5.1. This is an "augmented reality" photo, see Section 3.5.4.
Bottom right image: A user is interactively studying the air flow in and around the clouds using streamlines, see Section 6.3. This is an image from a playback of a Workbench session in the RWB Simulator, see Section 3.5.

About the back cover
The Virtual Workbench is being used for visualization of real-time Molecular Dynamics and steering of the simulation in the MolDRIVE system (Section 6.2).
Central image: A user at the Workbench is steering a particle of a protein using the spring manipulator, see Section 4.3.
Bottom left and right images:
Particle steering with the spring manipulator in an electrolyte simulation (left). The virtual particle steering method is assisted by a color slicer, which shows the particle potential around it (right).

Cover design by Michal Koutek

Scientific Visualization in Virtual Reality: *Interaction Techniques and Application Development*

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 12 mei 2003 om 10:30 uur
door

Michal KOUTEK

inženýr,

Fakulta elektrotechnická,
České vysoké učení technické v Praze

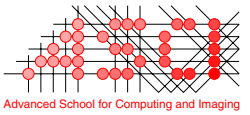
geboren te Praag, Tsjechië

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. F.W. Jansen

Toegevoegd promotor:
Ir. F.H. Post

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof.dr.ir. F.W. Jansen,	Technische Universiteit Delft, promotor
Ir. F.H. Post,	Technische Universiteit Delft, toegevoegd promotor
Prof.dr.ir. H.J. Sips,	Technische Universiteit Delft
Prof.dr.ir. J.J.van Wijk,	Technische Universiteit Eindhoven
Prof.Dr.rer.nat. B. Fröhlich,	Bauhaus-Universität Weimar
Prof.Ing. P. Slavík, CSc.	Czech Technical University in Prague
Dr.ir. A.F. Bakker,	Technische Universiteit Delft



This work was carried out in graduate school ASCI.
ASCI dissertation series number 85.

Published by:

Michal Koutek,
Computer Graphics & CAD/CAM group,
Faculty of Information Technology and Systems (ITS),
Delft University of Technology (TU Delft)

E-mail: M.Koutek@cs.tudelft.nl, Koutek@nat.vu.nl

WWW: http://visualization.tudelft.nl, http://graphics.tudelft.nl

Preface

The research described in this thesis was carried out in the Computer Graphics & CAD/CAM group at Delft University of Technology. The project was directly supervised by Frits Post. It is the sixth project in a series of PhD projects on data visualization, but the first project concerned with Virtual Reality and data visualization.

In summer 1998, the Responsive Workbench facility was installed at the High Performance Applied Computing Center (HPαC) at TU Delft. The Workbench was intended to serve as a high performance visualization system, working in a cluster with the other HPαC supercomputers.

This PhD project was initiated to set up an environment for high-performance data visualization, so that our group and other research groups of TU Delft could use this VR facility. Another aspect was to include computational steering facilities, which would enable the user to control a supercomputer simulation directly from the virtual environment displayed on the Workbench. For the purposes of our research we developed the RWB Library and the VRX toolkit, together a basic environment for visualization and interaction on the RWB.

The thesis covers three main topics: design and development of VR applications, interaction in virtual environments, and visualization of data, originating from scientific simulations. On various case studies we have demonstrated that the Responsive Workbench concept with our software and techniques can provide an efficient visualization environment with natural spatial interaction. The case studies were done in co-operation with internal TU Delft and external research groups. One of the early applications was an interactive 3D visualization of the flooding risk simulations, provided by WL|Delft Hydraulics. The Molecular Dynamics visualization and computational steering case study has been conducted in close co-operation with the Computational Physics group (Faculty of Applied Sciences, TU Delft). The visualization of atmospheric data, originating from cumulus clouds simulations, has been performed together with the Thermal and Fluids Sciences group (Faculty of Applied Sciences, TU Delft).

This thesis is accompanied by a CD-ROM that contains an electronic version of this thesis, video presentations for conferences, VR animations, images and web pages. It is strongly recommended to reproduce the CD-ROM and give it to everyone who is interested.

Many people have contributed to this research and were of importance for completion of this dissertation. I wish to thank them all.

First of all I would like to thank my direct supervisor Frits Post and my promotor Erik Jansen for giving me the possibility to conduct this research project in their group.

Frits, you are an extraordinary supervisor with a very broad spectrum of knowledge and a never-drying-out source of inspiration. You were a great teacher and advisor for me on the course of performing research and writing scientific papers. I have learned a lot from your very detailed reviews and corrections of my papers and this thesis. I am much obliged to you.

Erik, I want to thank you for your enthusiastic support through the project and also for constructive suggestions to this thesis.

I also appreciate very much Loek Bakker for his great ideas, technical support and keeping the Virtual Workbench operational; every 6 months some of the VR or graphics hardware got a failure. I remember that the fishing rod metaphor, which we used for particle steering, was your idea.

Next, I would like to thank all people at the CG & CC group and at the CP group: the (Ph.D.) students, the teachers, and the technical and administrative staff. They created a pleasant environment to work in, and an atmosphere which I enjoyed very much. I must also thank several M.Sc. students that I had the pleasure to work with during my PhD project: Gerwin de Haan, Jeroen van Hees, Jeroen den Hertog, and Michel Brinckman. We worked as a team together and we have learned a lot from each other. I will always remember the golden times when our VR lab was full of great people.

Further, I would like thank several colleagues from TU Delft, who provided us with their datasets or simulations that we have visualized on the Workbench: Harm Jonker - cumulus clouds simulation, Jaap Flohil - Gromacs simulation of proteins, Robert H.F. Chung - DEMMPSI simulation of electrolytes, and Guus Stelling - flooding risk simulation data. I also thank Anton Koning from SARA for giving us the possibility to test MolDRIVE in the CAVE. I should not forget to thank Anton Heijs who helped me at the beginning with the Workbench.

I am grateful to my former office mates: Freek Reinders, Eelco van den Berg, and Alex Noort for helping me to discover the mysteries of the Dutch language. Special thanks belongs to Freek for his patience in correcting my often wrong pronunciation (a typical example: *de mensen hebben "roest" nodig*).

I am thankful to VU Amsterdam, my new employer, in particular Henri Bal and Hans Spoelder, for giving me the possibility to finish this thesis.

Finally, my greatest thanks goes my wife Ilona for her love and unconditional support, and taking care of Sebastian, our little son, so that I could sleep in the night and fully concentrate on the writing of this thesis in the past months.

Delft, January 2003 Michal Koutek

Contents

Chapter 1

Introduction

The progress in computer graphics and virtual reality technologies in recent decades has made the Sutherlands' visions about "ultimate displays" become almost real [Sutherland, 1970]:

"We should look on the display as a window into a virtual world. Improvements of image generation will make the picture look real. Computers will maintain the world model in real time. Immersion in virtual worlds will be provided by new displays. Users could directly manipulate virtual objects. The objects will move realistically. Virtual world will also sound and feel real."

In a recent survey [Brooks, 1999] on the state of the art in Virtual Reality (VR), Brooks states that VR already works well but there is still much to be improved. Finding good VR applications, their implementation, and natural user interaction also belongs to the main VR issues, and these aspects were part of the motivation for this research.

Brooks defines a VR experience as any in which the user is effectively immersed in a responsive virtual world. VR displays and VR devices provide a user with interactive virtual worlds. The symbiosis between the VR hardware and the virtual world is usually called Virtual Environment (VE). It seems that VR is no longer an infant technology and has already found some serious applications. The increasing adoption of VR technology and its techniques is increasing productivity, improving team communication, and reducing costs.

The first and still the best VR applications are vehicle simulators, mostly for airplanes, cars or ships. Virtual prototyping, as an industrial VR application, is used by engineers to design, develop, and evaluate new products by fully using computer models. This branch is dominated by aircraft and automotive industries. VR applications in entertainment (games, virtual rides, interactive story telling, etc.) are traditionally also very successful. Among many other serious application areas we can mention architectural design, training of pilots and astronauts, military training and simulators, medicine (psychiatric treatment, surgical training and planning), and last but not least visualization of data from medicine, chemistry, pharmacy, geology, meteorology or other applied sciences.

Today's scientific simulations and data sensing/measuring systems produce enormous amounts of data. The only practical way after the statistical analysis, to get insight into "the numbers" of the simulated or measured data is to use data visualization. In recent years it has been demonstrated that Virtual Reality can also provide very natural environments with powerful techniques for visualization of scientific data [van Dam *et al.*, 2000; Brooks, 1999; Schmalstieg *et al.*, 1998; Dai *et al.*, 1997; Krüger *et al.*, 1995; Haase, 1994; Cruz-Neira *et al.*, 1993; Bryson & Levit, 1992].

Building an immersive visualization environment begins with a careful selection of a VR system. Before the beginning of this PhD project the Responsive Workbench was selected as a promising VR system for visualization of simulation results in computational science. We wanted to study in depth the aspects of developing VR applications, design of VEs, and user interaction with VEs. As application domain we have chosen visualization of scientific data. Indeed, we were expected to develop practical experience and knowledge about utilization of VR for our visualization group and HPαC. This PhD project was additionally initiated to set up an efficient environment for data visualization, so that our group and other research groups of TU Delft could use this VR facility as a useful tool on a more regular basis in our visualization projects.

1.1 Objectives

The main objective of the research described in this thesis was to study visualization of scientific data in Virtual Environments (VEs). We have developed new techniques for interactive data exploration and visualization on the Responsive Workbench (RWB), a projection-based Virtual Reality system.

For the purposes of our research we had to design and implement a basic software environment for visualization and interaction on the RWB. Further, we have also studied computational steering of remotely running simulations from virtual environments.

On various case studies we have proved that the Responsive Workbench concept with our software and techniques can provide an efficient visualization environment with natural three-dimensional interaction.

The objectives of this thesis are:

1. Implementation of development environment for RWB applications
2. Design of VEs
3. Interaction techniques for VEs
4. Visual force-feedback tools for manipulation of virtual objects
5. Steering of real-time simulations
6. Techniques and architectures for interactive visualization of data

The first objective, was to have a flexible software environment for the Responsive Workbench to facilitate rapid application development. For this purpose we have built the RWB Library and the RWB Simulator. This software environment was published in [Koutek & Post, 2001a, 2002]. With this basic software we could design VEs and develop new techniques for interactive visualization, and test them on various case studies.

A second objective was to have a coherent set of interaction methods for navigation in VEs, and for selection and manipulation of virtual objects. This thesis presents direct and remote interaction techniques for virtual assembly tasks. We have studied dynamic object behaviour during manipulation. Object constraints, collision detection, and collision behaviour were also considered. We have developed the spring-based manipulation tools to provide a visual force-feedback and to substitute the real force input. Research in this area was published in [Koutek & Post, 2000, 2001b, 2001c].

Another objective was to design and develop visualization and steering VE for remotely running real-time simulations. This thesis presents the MolDRIVE system, which provides visualization and computational steering of Molecular Dynamics (MD) simulations. The aspects and the techniques of particle (atomic) steering were published in [Koutek *et al.*, 2002]. In this paper we have presented an original particle steering technique: the Spring Particle Manipulator.

The final objective was to explore ways and techniques for interactive exploration and visualization of volumetric data in immersive VEs. To increase the intuitivity of the techniques we have employed two-handed interaction scenarios and made use of the pen-and-notepad metaphor. Suitable data abstractions were considered and fast probing tools were implemented into VRX, our visualization toolkit for VR. This research on interactive exploration tools for immersive VEs has been published in [de Haan, Koutek & Post, 2002].

A number of M.Sc. students has participated in several research areas of this thesis. The MolDRIVE system has been developed in a team co-operation [van Hees & den Hertog, 2002; de Haan, 2002]. Van Hees and den Hertog worked mainly on implementation of an interface between MD simulations and VEs. De Haan worked on visualization techniques of MD data in VEs. De Haan has also significantly contributed to the development of the interactive exploration techniques and the VRX toolkit, which was successfully used in the third case study (Section 6.3). Further, Michel Brinckman has worked on the user-assisted tracking of clouds in the third case study [Brinckman, 2002].

1.2 Structure of This Thesis

Chapter 2 provides an overview of related work in the field of visualization in VR. In Section 2.1 general principles of scientific visualization are described. Section 2.2 gives an overview on the state of the art in Virtual Reality and VR systems. Further, Section 2.3 discusses the research issues of scientific data visualization by means of VR. Finally, Section 2.4 describes motivation of our work and outlines the research agenda of this thesis, which has been derived from the general research issues of VR and visualization in VR.

Some sections of the following chapters are mainly based on our recently published work. Chapter 3 and especially Sections 3.4 and 3.5 were based on two conference papers [Koutek & Post, 2001a, 2002]. Section 4.2 was created as a conjunction of three papers [Koutek & Post, 2000, 2001b, 2001c]. Section 4.3 has been published in [Koutek *et al.*, 2002]. Section 5.1 is also based on one paper [de Haan, Koutek & Post, 2002]. As we wanted to keep the text of those sections as fluent and compact as possible, some overlap in text and figures with the case-studies chapter could not be avoided.

Chapter 3 describes the visualization concept of the Responsive Workbench. Sections 3.1 and 3.2 give an introduction and an overview of the technical aspects of the RWB. In Section 3.3 the focus is on the design aspects and issues of VEs for visualization on the RWB. Section 3.4 presents the RWB Library and the RWB Simulator.

Chapter 4 deals with interaction in VEs. Section 4.1 gives an overview of the basic interaction techniques in VEs and presents a basic interaction set suitable for the Responsive Workbench. The VR aspects of object collisions and object constraints are also discussed. Section 4.2 presents visual force-feedback tools which are based on the spring metaphor. Particle steering tools of MolDRIVE, our Molecular Dynamics visualization system, are explained in Section 4.3.

Chapter 5 describes our approach to interactive visualization and exploration tools for VR. In Section 5.1 a set of intuitive interaction and exploration tools for VEs is demonstrated. Section 5.2 presents VRX, our modular object-oriented toolkit for exploratory data visualization.

Finally, validation and application of the visualization concept and VR techniques described in this thesis have been performed on several case studies, as described in Chapter 6.

It contains three main case studies:

- Interactive visualization of flooding scenarios (Section 6.1)
- Molecular Dynamics visualization and computational steering (Section 6.2)
- Visualization of cumulus clouds (Section 6.3)

At the end of each chapter conclusions are given, followed by a discussion on future work in the topics concerned in the chapter. Chapter 7 presents overall conclusions and gives directions for future research.

Chapter 2

VR in Scientific Visualization

2.1 Scientific Visualization

"Scientific visualization is the use of computer graphics to create visual images which aid in the understanding of complex, often massive numerical representations of scientific concepts or results [McCormick, 1987]." Such numerical representations, or datasets, may be output of numerical simulations as in Computational Fluid Dynamics (CFD), Molecular Dynamics (MD) or engineering in general, sensing (recorded) data as in geological, meteorological or astrophysical applications. In case of medical data (CT, MRI, etc.) we usually use term medical visualization.

Visualization is essential in interpreting data for many scientific problems. It transforms numerical data into a visual representation which is much easier to understand for humans. Other tools such as statistical analysis may present only a global or localized partial view on the data.

Visualization is such a powerful technique because it exploits the highly skilled human vision (more than 50 percent of our neurons are devoted to vision). While computers excel at simulations, numerical operations, data filtering, and data reduction, humans are experts at using their highly developed pattern-recognition skills to look at anomalies. Compared to programs, humans are especially good in seeing unexpected and unanticipated emergent properties [van Dam, 2000]. The human eye has phenomenal capabilities for detecting structures, shapes and patterns.

"Scientific visualization is not an end in itself, but a component of many scientific tasks that typically involve certain combination of interpretation and manipulation of scientific data and models. To aid understanding, scientists visualize the data to look for patterns, features, relationships and anomalies. Visualization should be thought of as task driven rather than data driven [van Dam, 2000]."

Simulation and visualization are used as an alternate means of observation, creating hypotheses and testing the results of simulations against data from physical experiments. Simulations may use visualization as a separate post-process or may interlace visualization and parameters setting with re-running the simulation. We speak then of *computational steering* [McCormick, 1987], in which the user monitors and influences the computation process. Computational steering closes the loop such that the scientists can respond to results of the simulations as they occur by interactively manipulating the input parameters of the simulation. This technique enhances productivity by greatly reducing the time between changes of parameters and viewing of the results. Brooks ex-

pressed a need for generalized tools for interactive steering of large computer simulations [Brooks, 1988]. Over the years, many computational steering applications and systems have been developed. An overview of computational steering environments, such as VASE, SCIRun, Progress & Magellan, SMD, CUMULVS, and CSE, is given in [Mulder *et al.*, 1999].

Traditionally, scientific visualization has been used in two modes: *exploration and presentation.* Goal of data exploration is to find relevant features or patterns in the data, that contain the studied phenomenon. During this process the user is manipulating visualization techniques and changing a view on the data. When an appropriate view on the feature/aspect of the data is discovered, then scientific visualization can produce static images or animations for presentation of the the investigated phenomenon.

The process of data visualization can be described as a sequence of fundamental processing steps [Haber & McNabb, 1990], *the visualization pipeline:*

- **Simulation:** results of numerical simulations (or data sensing / measurement) are the input of the visualization pipeline.
- **Data selection & filtering:** relevant regions of the raw data are selected, then filtered and enhanced. Techniques such as: i.e. enrichment & enhancement, data cropping, down-sizing, noise filtering, segmentation and feature extraction can be used.
- **Visualization mapping:** the processed data have to be mapped / transformed into graphical primitives such as: i.e. points, lines, planes / surfaces (triangle meshes), or icons, and their properties such as: color, texture, or opacity.
- **Rendering:** finally, the graphical primitives are rendered as images, which are then displayed on the screen.

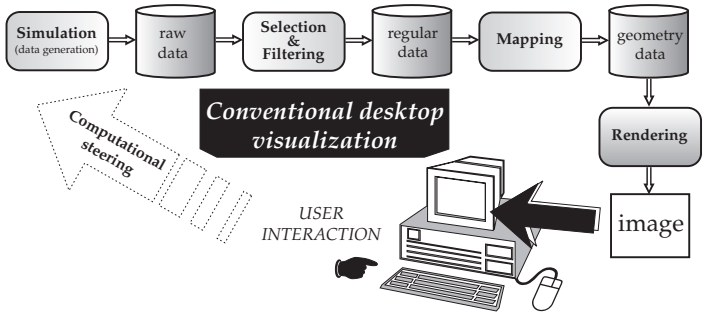


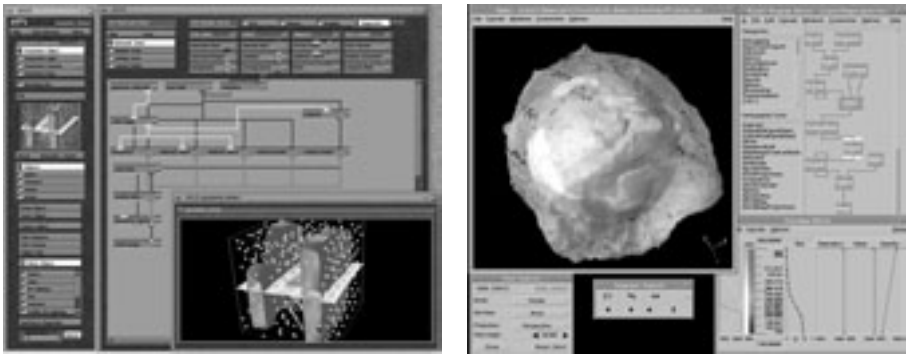
Figure 2.1: Visualization pipeline on desktop workstations

We should think of the visualization as an interactive process, especially in the exploration phase. On conventional desktop workstations (see Figure 2.1) the users are provided with the keyboard and mouse interface and through the

visualization systems they can usually interact with the stages of the visualization pipeline. If a visualization system provides a good interactive control over the processing elements (not only changing a view on the data in the rendering module, but also changing visual representations, filtering parameters, etc.), then it can become a powerful research tool.

In the early days of visualization, it was rather difficult for the researcher to visualize data beyond conventional drawings and plots. Familiarity with computer graphics programming was required for more sophisticated 3D visualization. In the recent decade(s), the problem of using visualization as a research tool by scientists (non-graphics experts) has been addressed through the development of powerful visualization systems which offer the user visualization capabilities without requiring programming skills. There are basically two approaches: *specialized visualization programs* (Vis5D [Hibbard & Santek, 1990], VMD [Humphrey *et al.*, 1996], and many others) and *general purpose data-flow visualization systems,* see Figure 2.2.

The latter approach can be divided into: *visualization network editors* (AVS [Upson *et al.*, 1989], OpenDX [Lucas *et al.*, 1992], Iris Explorer [Foulser, 1995]) and *visualization programming interface libraries* (VTK [Schroeder *et al.*, 1999]).



(a) AVS (b) OpenDX

Figure 2.2: Data-flow visualization systems working as network editors.

Visualization network editors (application builders) are very popular. By simply connecting visualization modules, represented by graphical icons, a network can be built and the data flow through the network can be defined. Each of the modules in the constructed network can have its unique functionality, manipulating the data that flows through the module. Although these modules can be customized for a specific application, there are numerous general purpose modules, representing useful functions such as reading data, creating and rendering geometry. The user can select a data reader module from the library of input modules, drag it to the network editing area, connect its output to a visualization mapping module and connect its output to a *geometry viewer* module. The user can control parameters of several modules in the pipeline

and navigate in the visualization window to get the best view on the data. The network builder approach should provide a simple user-friendly and effective solution for researchers to get a view on their data.

Another approach is to construct the visualization pipeline completely in a programming language (e.g. Java, C++) or a scripting language (e.g. Tcl, Python) using a visualization programming library such as the Visualization ToolKit (VTK) [Schroeder *et al.*, 1999]. VTK is a programming library with numerous visualization modules that can be customized or used directly in almost any visualization application. Although this approach is more difficult to use and is not very popular with end users, it gives developers the possibility to use the visualization tools in their own application development environment.

A collaborative visualization environment such as COVISE [Rantzau *et al.*, 1998] allows multiple scientists to study visualizations of their data collaboratively via a network, see Figure 2.3.

Figure 2.3: COVISE - collaborative visualization [Image source: RUS]

During visualization on a desktop workstation a window is provided to visualize the results of the pipeline. The user interface plays an important role. It has to offer interaction methods for efficient data exploration. In a regular desktop visualization this control is often supplied by a Graphical User Interface (GUI), using buttons, menus and widgets to adjust parameters of the visualization pipeline. Most users are quite familiar with this 2D interface.

However, performing interactive 3D visualization on regular 2D screens (using perspective projection of 3D space onto a 2D window) can hide essential spatial features of the data. In addition, it is hard to navigate through the data and spatially control the orientation and position of the various visualization tools using the classical 2D user interface. The ambiguous control of the visualization environment sometimes hinders effective exploration of the datasets.

Virtual Reality has the potential to enhance both display of 3D graphics and spatial control, offering a better environment for exploration. The following sections describe Virtual Reality and its application in visualization.

2.2 Virtual Reality

The original term Virtual Reality (VR) refers to any computer generated 3D environment (VE), in which a user is practically immersed. *Immersion* can be characterized as an experience of being enveloped by, included in, and interacting with the VE. It is not required that VEs faithfully mimic the real world. As there are different levels of user immersion, the term VR is sometimes used in a confusing and misleading manner. Today the boundaries between interactive 3D computer graphics and VR have blurred. Applications like photo-realistic computer-generated movies, flight simulators, 3D action games and desktop 3D worlds are often placed in the class of Virtual Reality. Similar and related terms to VR include Synthetic Environments, Artificial Reality, Cyberspace, Virtual Worlds and Virtual Environments.

2.2.1 VR Definition

Virtual reality is the use of computer technology to create the effect of an interactive 3D world in which the objects have a sense of spatial presence. The primary difference between conventional 3D computer graphics and VR is that in Virtual Reality we are working with things instead of pictures of things [Bryson, 1994a].

In this thesis, we refer to the concept of immersive Virtual Reality, which gives the user the psycho-physical experience of being present in a virtual environment consisting of interactive (virtual) objects. This experience is achieved by a proper integration of VR hardware (3D displays and spatial interaction devices) with a responsive computer-generated 3D environment. (Note that VRML applications are typical examples of non-immersive VR.)
[Brooks, 1999; Burdea & Coiffet, 1994; Durlach & Mavor, 1995] have summarized four technologies which are crucial for VR:

- the visual (and aural/acoustic and haptic) displays that immerse the user in the virtual world and that block contradictory sensory impression from the real world;
- the graphics rendering systems that generate at least 20-30 stereo images per second for each eye;
- the tracking system that periodically reports the position and orientation of the user's head and limbs;
- the database construction and maintenance system for building and maintaining of the virtual world model.

The auxiliary technologies are also important, but not so crucial:

- synthesized sound, including directional sound and sound effects;

- synthesized forces and other haptic sensations to the kinesthetic senses;
- realistic behaviour of objects in VEs;
- improved interaction devices, interaction techniques that substitute for the real world interactions.

The sensation of space and depth is essential for every VR system. The human visual system interprets the depth in sensed images using both physiological and psychophysical cues [Okoshi, 1976]. The *physiological depth cues* are accommodation, convergence, binocular parallax, and monocular motion parallax. Convergence and binocular parallax are the only binocular depth cues, all others are monocular. The *psychophysical depth cues* are retinal image size, linear perspective, texture gradient, overlapping, aerial perspective (fog/haze), shading and shadows. In the real world people use all available depth cues to determine distances of objects and their spatial relations.

Through the use of artificial depth cues in computer graphics, these spatial sensations can be simulated. In regular desktop 3D graphics, monocular depth cues such as perspective, shading, shadows and texture gradients are often used. In IVR, the *stereo display* and *head tracking* are used to also provide the *binocular parallax* and *motion parallax*, respectively.

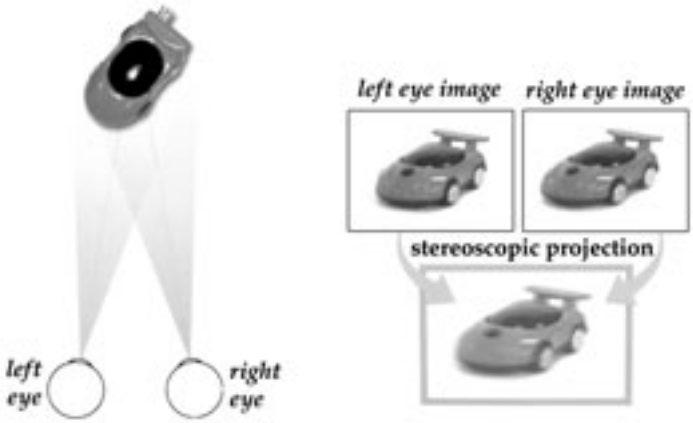
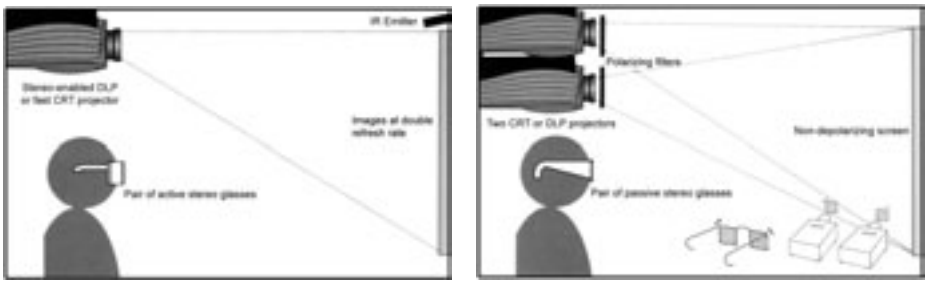


Figure 2.4: Binocular parallax and stereo effect [Image source: Barco]

The *binocular parallax (or stereopsis)* is achieved by displaying a separate image for each eye (stereo display). The human visual system uses the slight differences in the images to reconstruct depth information. To simulate this strong depth cue, the computer generates two images from slightly different viewpoints on the VE, see Figure 2.4. These two images are displayed to the left and right eye separately. The images are mentally merged by a human observer, providing the 3D depth / stereo effect. The separation of images can be achieved by many techniques, of which passive stereo (using polarized light projection) and active stereo are very popular, see Figure 2.5.

In **active stereo,** the two images for the left and right eye are alternately projected at a high frequency (e.g. 2*48 = 96 Hz, or 2*60 = 120 Hz). At the same frequency these left and right eye images are switched, the LCD shutter glasses obscure light directed to one of the eyes (Figure 2.5(a)). As a result, each eye of the user only sees the images intended for that eye.



(a) (b)

Figure 2.5: (a) Active stereo display using one projector and shutter glasses. (b) Passive stereo display provided by polarized light from two projectors and polarized glasses (linear polarization is shown, but circular polarization may be also used). [Image source: Barco]

In **passive stereo,** the two images intended for the left and right eye are projected by two separate projectors on a silver screen. Polarizing filters are mounted on these projectors, while the users wear polarizing glasses. The polarization direction of the filters on both the projectors and the glasses is perpendicular (Figure 2.5(b)). In this way the user only sees the image generated by one projector on the left eye and the other image on the right eye.

Figure 2.6: Autostereoscopic display with optical camera-based eyeball tracking [Image source: Dresden3D]

Recently, the *autostereoscopic displays* (i.e. Dresden3D [Web-Dresden3D]) were developed and brought to the market. These displays present a spatial image for a viewer without using glasses, goggles, or other viewing aids, see Figure 2.6.

However, these displays still suffer from a limited display resolution and other technical problems. A survey on autostereoscopic and holographic displays can be found in [Halle, 1997].

Head tracking is used to simulate motion parallax and to measure the spatial position and orientation of the user's head. It interactively controls the viewpoint on the virtual world, from which the images are generated. To provide a non-distorted VR experience the rendering system must deliver at least 10 frames per second [Bryson, 1994a].

Immersion in the VEs is typically produced by a stereo 3D display, which uses head tracking to create a human-centric rather than a computer-determined point of view on the virtual world. Basically, there are three types of VR displays: *head-mounted displays (HMD),* which have small display screens in front of the user's eyes, *projection-based displays* (CAVEs, workbenches, panoramic displays), which are specially constructed rooms, walls or tables with stereo projection, and *personal VR displays,* usually integrated within desktop VR systems.

Presence: closely related to the sensation of immersion is the sensation of presence. It can be described as the feeling of "being in the same space as the VE", which gives a sense of the reality of objects in the computer-generated scene and the user's presence with those objects. Both immersion and presence are enhanced by a wider field of view than is available on desktop displays. This helps to provide situation awareness, aids spatial judgements, and enhances navigation and locomotion. The VR experience can be enhanced with 3D sound and by haptic devices, which provide touch and force feedback, see Section 4.2.

Interaction with the VE is provided through a variety of spatial input devices, most of these working in 6 degrees of freedom (DOF) based on tracking technology. Such devices include *3D mice*, various kinds of *wands* (or interaction stylus) with buttons for pointing and selecting, *data gloves* that sense joint angles, and *pinch gloves* that sense fingertip contacts. Both types of gloves provide position and gesture recognition. Additional sensory modalities are exploited in speech recognition and haptic force input. More about interaction with virtual environments, including related work, can be found in Chapter 4.

2.2.2 Head Mounted Displays

The classical way of providing immersion of users in virtual worlds is through the Head Mounted Displays (HMD). The user is wearing a helmet in which two small LCD screens and a head tracker are mounted, see Figure 2.7. By looking at the screens and moving around, the user is completely surrounded by the virtual world and is visually isolated from the real environment.

Since 1996 see-through HMDs are also available, in which the users can also see objects of the real world and their own limbs. With classical HMDs, the user's limbs had to be modeled and placed into the virtual world. HMDs have been significantly improved in image resolution, color saturation, brightness, weight and ergonomics. But still most of these parameters do not completely

meet the user's needs. The light-weight HMDs still have a rather limited field of view (45 degrees). Nevertheless, HMDs are very popular VR displays, and they are used in many types of applications, including scientific visualization, see Figure 2.18. In applications where HMDs do not work well the VR users tend to employ the projection-based displays.

(a) (b)

Figure 2.7: (a) A conventional HMD; (b) Fear of heights phobia treatment using HMD [Image source: M.J. Schuemie, ITS TU Delft].

2.2.3 Projection-based Displays

This technology displays the stereo images via CRT, DLP, or LCD projectors onto a wall, a foil or a frosted glass. It offers much larger field of view on the displayed virtual world with higher image resolution (more than 1024x768 per projector), and usually better image quality than HMDs. More users can be immersed at the same time. Also the ergonomic aspects seem better.

Although the projection-based systems are often more expensive than an HMD, it is generally believed that the better display quality and their usability value are worth their price. Projection-based systems can be implemented in two ways: *in active stereo* or *passive stereo,* with rather expensive shutter glasses or much cheaper polarizing glasses, respectively. Between shortcomings of these systems belong: the limited brightness, contrast and sharpness of the projected images, problems with focus, often problematic calibration and alignment of projectors in a multiple-projector setup, and finally the need to work in darken room. They need a proper installation and utilization for optimal results.

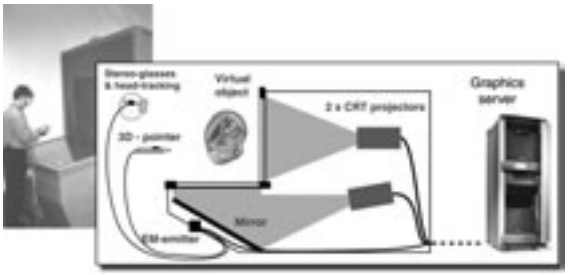
Virtual Table Systems

Classical virtual table systems or workbenches can be built either with one projector in active stereo or two projectors in passive stereo. These systems are represented by ImmersaDesk (Figure 2.8(a)), BaronTable (Figure 2.8(b)), and the Responsive Workbench originally developed by [Krüger *et al.*, 1995]. For a detailed description of this concept see also Section 3.1.

The Workbench systems provide partial immersion into the displayed virtual world. In fact the virtual and real worlds coexist. For applications that use the laboratory table metaphor, it is seen as an advantage that the user remains present in the real world instead of being fully immersed in a VE.

(a) ImmersaDesk [Image source: Fakespace]

(b) Baron [Image source: Barco]



(c) Holobench [Image source: GMD, TAN]

(d) Consul [Image source: Barco]

Figure 2.8: Virtual table systems

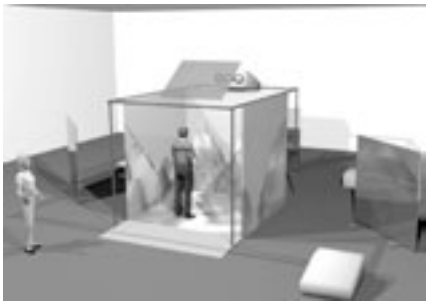
Since the eye-to-far-screen-edge plane limits the apparent height of virtual objects, many workbenches can be tilted to resemble drafting tables,see Figures 2.8(a) and 2.8(b). This problem has been also addressed with the Holobench concept which has an extra vertical screen, see Figures 2.8(c) and 2.8(d).

Workbench applications are in GIS (geographic information systems), architecture, chemistry, medicine, modeling and virtual prototyping. The Workbench systems excel in visualization of complex three-dimensional data.

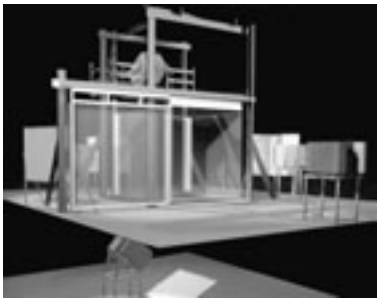
Fully Immersive VR Systems

The original concept of the CAVE was developed at EVL, University of Illinois [Cruz-Neira *et al.*, 1993], see Figure 2.9(a). Originally, this VR system provided surround-screen projection on four sides (left, right, front, and ground). The projection screens were driven by a set of coordinated image-generation systems. CAVE-like systems have been later implemented also by others under different names: VR-CUBE, Cyber Stage, I-SPACE, HyPI6, RAVE, ReaCTor, C2, or Reality CUBE. The CAVE systems provide complete immersion in the virtual worlds by using either passive or active stereo. In active stereo, the 4-sided CAVE is built up with four projectors (usually CRT).

Recently, in 2001 a 6-sided CAVE called HyPI6 was built at Fraunhoffer IAO in Stuttgart, see Figure 2.9(b). They have installed 12 rendering PCs with Linux, OpenGL and Performer for 12 projectors as an alternative to the multi-processor and multi-pipe SGI Onyx2. HyPI6 can work either in active stereo (only 6 projectors used) or in passive stereo, using all 12 projectors. As the back wall (sliding door) closes the space, wireless tracking and wireless audio have to be used. This system provides a complete surrounding projection. A serious issue is that the user can very easily lose the orientation inside; as the relation to the real world is almost completely lost.



(a) 4-sided original CAVE [Image source: EVL]

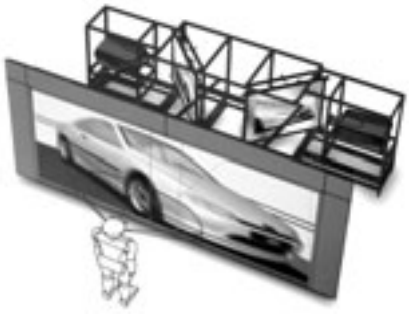


(b) 6-sided HyPI6 [Image source: Fr. IAO]

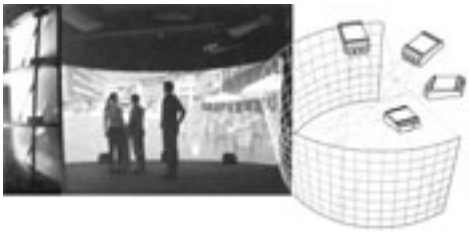
Figure 2.9: Fully immersive CAVE-like systems

Virtual Wall and Panoramic Display Systems

Certain industrial applications require displaying objects in their real-world size. Objects like cars do not fit into the CAVE-like environment (3x3x3m). Automotive industry thus demands large panoramic tiled displays. Many types of tiled wall-projection systems have been developed, including: Tanorama CYLINDER & PowerWall, CADWall, WorkWall I-CONE, or IC-Wall.



(a) CADWall [Image source: Barco]



(b) I-CONE [Image source: Fraunhoffer IMK/GMD]

Figure 2.10: Virtual panoramic wall systems

In Figure 2.10 two examples are shown: (a) large tiled projection wall for displaying life-size models of cars, (b) panoramic cylindrical projection for urban and architectural walk-through.

As these projection systems offer very large display areas and higher resolution of images (number of projectors times 1280x1024), they are also very useful for visualization of large scientific datasets. These systems can be built in both active and passive stereo; even monoscopic projection on such a large (sometimes also panoramic) wall can provide a high level of immersion.

Projection-based displays can be customized and used together with a car mockup for a driving simulator, see Figure 2.11(a). The user inside the car has a panoramic view on the road (three back-projected screens in front), side view on the left (one back-projected screen on left side), and three mirror views (three front-projected screens behind the car).

(a) Immersive driving simulator [Fr. IAO] (b) Personal Immersion [Fraunhoffer IAO]

Figure 2.11: Customized (multi-) wall projection systems

Due to the increasing performance of PC processors and graphics processors, such a customized display system can be built with a cluster of Linux PCs, which seems to be a current trend for low-cost affordable VR systems. In Figure 2.11(b) is shown a simple example of one projection wall driven by two PCs.

2.2.4 Personal VR Systems

The large and expensive VR systems, such as CAVEs and panoramic walls, are not affordable or suitable for the daily-use in real research work in many application areas [Poston & Serra, 1996; von Wiegand *et al.*, 1999; Mulder & van Liere, 2002]. Certainly, it is nice when the visualization results of research studies can be presented in larger (fully) immersive VEs to a larger audience. But many applications i.e. from chemistry, biology, and medicine do not explicitly require immersion of the users in large VEs.

What is required are ergonomic desktop VR systems, with which the scientists can work longer than 30 minutes in their natural environment (own laboratory), in normal lighting conditions. These personal VR systems should be of compact desktop size, inexpensive and affordable. This seems to be the best way to make VR technology accessible and affordable for the real daily use in scientific and clinical practice.

Figure 2.12: The user can manipulate the protein model with his left hand, holding a cube with markers. The protein geometry can be clipped by using a clipping tool in the right hand. [Image source: CWI (Netherlands)]

Mulder & van Liere have presented the Personal Space Station (PSS), a near-field virtual environment, addressing the issues of direct interaction, ergonomics, and costs [Mulder & van Liere, 2002]. In a mirror the user can see stereo images, which are generated by a cluster of two Linux-based PCs. Head tracking is realized by ultrasound trackers.

The PSS uses optical tracking for user interaction. Two cameras sense the interaction space, and computers analyze the images, tracking visual markers on interaction tools, see Figure 2.12. Advantage of optical tracking is that the user can naturally interact with the VE by using real instruments.

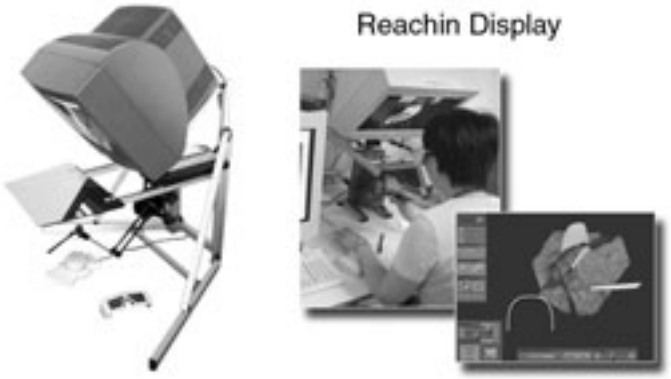


Figure 2.13: Reachin display with a Phantom haptic device provides also force feedback. [Image source: Reachin]

Another commercially available desktop VR system is the Reachin Workstation [Web-Reachin], see Figure 2.13. The system is equipped with a Spacemouse for navigation and a Phantom haptic device allowing 3D interaction, and providing force feedback.

Similarly, Poston & Serra have developed the Dextroscope, a kind of virtual workbench on the desktop [Poston & Serra, 1996]. It uses electro-magnetic tracking and a stylus for interaction. The Dextroscope [Web-Dextro] is very carefully designed and fulfills the ergonomic requirements, see Figure 2.19. This system is commercially available for medical applications.

2.2.5 Virtual Reality: Research Issues

VR is a logical extension of interactive 3D computer graphics and several of their research issues overlap. However, VR is a special case. Its research agenda incorporates the development and application of VR display and interaction technology. A large amount of research is also conducted in the area of Human Computer Interaction (HCI). As the interaction with VEs is very different from interaction with a computer, the VR issues go much beyond the conventional HCI problems.

VR Requirements and Research Agenda

In earlier years the performance requirement was that the scene has to be re-rendered from the current user viewpoint at least 10 frames per second (fps), so that the immersion effect would not be lost. Today we wish to see the virtual worlds displayed rather at higher frame rates 20-30 fps.

Engineering and architectural applications require a rather high level of visual realism of the virtual objects. Scientific visualization does not generally require realistic rendering. Nevertheless, there is a great need to render massive geometric scenes. And it challenges the graphical hardware, which is currently technologically driven by the game industry. The current graphics processors (GPU's) have rendering throughput of geometric primitives in order of 10M triangles per second, which is not sufficient. In the near future it should be about 300M triangles.

An important aspect is the *system latency*. It is required that the system response to a user input must occur within 0.1 seconds. Longer delays result in a significantly degraded ability to control objects in the VE [Bryson, 1994a].

Failure to meet the performance requirements will cause the immersion to fail, with consequences for the usage of such system. It can cause people to feel very uncomfortable in the VE, a condition sometimes called *cybersickness*. Everything required to support the working of the virtual environment, including data management and access, user interaction, computation, and rendering, must take place within these performance constraints. It remains a challenge for many researchers to meet these criteria for scientific visualization in VR.

We can summarize the general VR research issues as follows:

- Improved display technologies (new light production technologies, new display surfaces, high resolution, high luminance, constant color, and automatic calibration of multi-projector environments)
- More realistic rendering, maximize rendering speed and minimize latency
- More accurate and wide-range tracking (optical, acoustic, electromagnetic)
- Improved interaction with VEs (improved devices, gesture recognition, intuitive interaction metaphors)
- Implementation of VEs, model aquisition, and virtual world maintenance
- User friendly and ergonomic design of VR systems
- Acoustic and haptic augmentation of VEs
- Finding production-stage applications and choosing best fitting displays
- Cope with the lack of standards (or create them)

Of course, not all of these general issues could be addressed by this thesis, but they show a wider scope of problems in VR, and in a certain sense they do place accents in our work.

[Brooks, 1999] still sees the end-to-end system latency as the most serious technical constraint of today's VR systems. In HMD systems, head rotation is the most demanding motion, with an angular velocity of about 50 degrees per second. Latency of 150-500 ms makes the scene "swim" for the user, seriously reducing the presence effect. Latency is extremely serious in *augmented reality systems* in which the virtual world is superimposed on the real world. In projection-based VR systems the head rotation is related only to a viewpoint translation. Small head rotation means a small translation of the eye-points, thus also a small difference in the stereo images, see Figure 3.5 in Section 3.2. While the user does not move the head very fast, the system latency *(head tracker updating the view on the scene, rendering of the scene, and display)* is not so noticeable. In projection-based VR systems 150-250 ms of latency is generally accepted.

Interaction with VEs is a big issue of the current and future research. Related work in this research area and our contribution to improving interaction are discussed in detail in Chapter 4.

Lack of standardization of VR technology and software interfaces makes it difficult to apply VR on a larger scale. Not only developing VR applications is an issue, but also finding serious production-stage application seems to be still rather difficult. VR for visualization forms still a very small, specialized, and expensive market. Therefore, most of the VR hardware and tools suitable for visualization are often initially developed by the scientific community itself. Due to steady progress in the mass market of CPU's and GPU's the costs of VR systems are no longer dominated by the price of the computer, but by relatively expensive VR displays and trackers. Virtual Reality needs good applications and the VR technology must become affordable.

2.3 Visualization in VR

There are several reasons why (immersive) Virtual Reality can provide a good environment for scientific visualization [Bryson, 1994b; Haase, 1994]. The data are often high-dimensional, and are represented in a three-dimensional (3D) volume. Visualization of the phenomena associated with this data often involves 3D structures. Shapes and relations of 3D structures are often extremely important. VR can display these structures, providing a rich set of spatial and depth cues. Further, VR interfaces allow rapid and intuitive exploration of the volume containing the data, enabling the various phenomena at different places in that volume to be explored. Scientific visualization is oriented towards the informative display of abstract quantities, instead of attempting to realistically represent objects of the real world. Thus, the graphics demands of scientific visualization are oriented towards accurate, as opposed to realistic, representations. Graphical representations can be chosen which are feasible with current technology. Further, as the phenomena being represented are artificial, a researcher can perform investigations in VEs which are impossible or meaningless in the real world.

In many ways the main impact of VR technology on scientific visualization is in providing an intuitive and responsive interface for exploration of data. To achieve a maximal benefit, the VR visualization can be integrated within a computational steering environment, providing a virtual laboratory, where the researcher can instantly visualize the data and interactively steer the simulation.

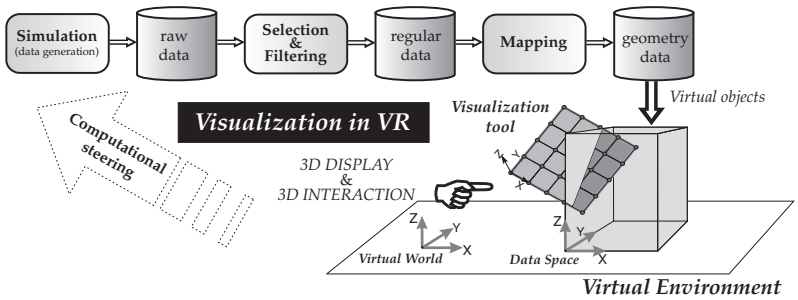


Figure 2.14: Visualization pipeline integrated with a responsive VE.

Visualization on desktop workstations produces static or animated images, see Figure 2.1. Interaction with the visualization pipeline is usually provided via 2D user interfaces (keyboard and mouse). Visualization in virtual environments produces interactive virtual objects instead of images, see Figure 2.14. These virtual objects are present in the VE, and can have customized look and behavior. The objects may be directly selected (touched) and manipulated by the user. The proper employment of immersive VR techniques and VEs can revolutionize the way the data are visualized and how people operate visualization environments. Instead of displaying 3D worlds inside of computer monitors

with conventional 2D interaction, the scientists can now see these 3D worlds in the same space where they can naturally interact with the virtual objects.

The evolution step from desktop visualization to visualization in VR is more radical than it might seem. When designing and developing VEs for data visualization we cannot just simply think of generating high quality graphical primitives and rendering nice images. The *"immersive visualization"*, as it is sometimes called, is a very complex and comprehensive problem, which has been challenging for researchers in the past years. A recent progress report on scientific visualization in VR [van Dam *et al.*, 2000] sketches the research agenda and its authors also "call to action", to help the scientist to employ VR technology and incorporate it into the scientific work-flow.

Unfortunately, computational requirements and dataset size in science research are growing faster than improvements in storage, networks, and processing power (Moore's law). The main bottleneck continues to be the ability to interactively visualize the large data and gain insight. While the raw polygon performance of graphics cards may keep up with Moore's law, visualization environments are not improving at the same rate. The key barriers to achieving really effective visualizations are underpowered hardware, underdeveloped software, inadequate visual encoding representations, interaction which is not based on a deep understanding of human capabilities, and a limited funding for visualization [van Dam *et al.*, 2000].

Short term solutions may include parallel visualization clusters, tiled displays (increased image resolution), and VR should help the most. Long term solutions will be probably based on artificial intelligence techniques to cull, organize, and summarize the raw data prior to VR viewing, while ensuring that the links to the original data remain. These techniques will support adjustable detail-and-context views to let researchers zoom-in on specific areas while maintaining the context of the larger dataset.

VR is used in scientific visualization in two kinds of problems: human-scale and non-human scale problems. The use of VR is obvious for vehicle simulators, vehicle design, and architectural design, as summarized in [Brooks, 1999].

For example an architectural walkthrough will, in general, be more effective in VEs than in a desktop environment because humans have a lifetime of experience in navigating through, making spatial judgements in, and manipulating 3D physical environments. Ergonomic validation tasks, like checking viewable and reachable cockpit instrumentation and control placement, can be performed more quickly and efficiently in a virtual prototyping environment than with labor-intensive physical prototyping.

The key question for non-human (micro/macro) scale problems is whether the added richness of life-size immersive display allows faster and easier work. So far, not enough controlled studies have been done to answer this question definitively. Anecdotal evidence indicates that it is easier to do for example molecular docking for drug design in VR than on the desktop [Haase *et al.*, 1996]. Direct interaction is impossible in physical experiments, but possible in VR.

In addition, we often have to visualize data that have no inherent geometry (such as flow field data) and perhaps no physical scale (such as statistical data). The 3D abstractions through which we visualize these datasets often present very irregular structures with complicated geometries and topologies. Just as VR allows better navigation through complex architectural environments, many researchers believe that VR is also a good environment for navigation and exploration of any complex 3D structure [van Dam *et al.*, 2000].

Virtual Reality naturally encourages collaboration of scientists in the process of data visualization and gaining insight. The participants do not necessarily need to be at the same location. *Telecollaboration / tele-immersion* allows multiple participants to interact with a shared dataset and between one another over a network. The collaborative work also incorporates teleconferencing techniques. Examples of tele-immersive collaboration in the CAVERN (CAVE Research Network) were presented in [Johnson & Leigh, 2001].

2.3.1 Example Visualization Applications in VR

[Bryson, 1994b] made the case that real-time exploration is a desirable capability for scientific visualization and immersive VR greatly facilitates the exploration process. Bryson's pioneering work on the Virtual Windtunnel (VWT) lets researchers study a fluid flow around a space shuttle [Bryson & Levit, 1992].

The visualization techniques could be controlled via direct manipulation or via widgets. In the VWT several techniques are available such as: sample points, streamlines, particle paths, iso-surfaces, color-mapped cutting planes, tufts, and numerical display, see Figure 2.15.

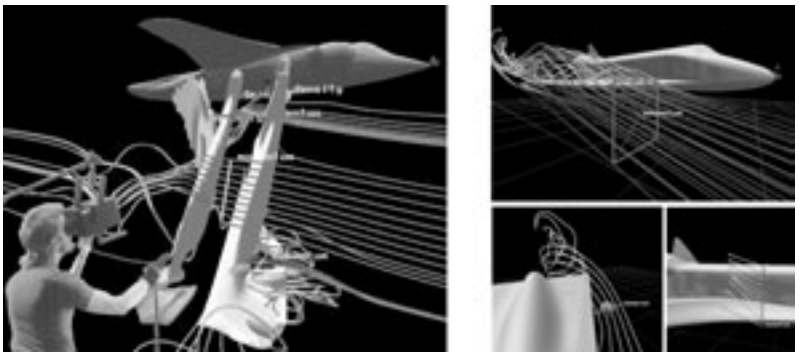


Figure 2.15: Virtual Windtunnel (VWT) application implemented with the Fakespace BOOM display and the VPL Dataglove [Image source: S. Bryson, NASA AMES]

When first presented the VWT was revolutionary in the sense that it allows investigation of flow fields in VEs at reasonable frame rates. It has been a great source of inspiration for many researchers in this area.

Another milestone of the VR-visualization history was the development of the projection-based VR displays, such as the CAVE [Cruz-Neira *et al.*, 1993] and the Responsive Workbench [Krüger *et al.*, 1995].

(a) Molecular modeling and docking (b) Fluid dynamics visualization in VWT

Figure 2.16: Visualization on the Responsive Workbench. [Image source: Krüger et al., Fraunhoffer IMK / formerly GMD]

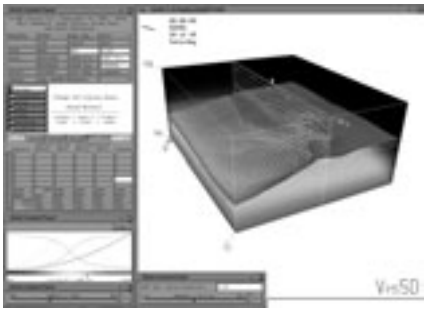
The Workbench concept became popular and has been adopted by many visualization groups for scientific research. The early Responsive Workbench applications have demonstrated the great potential of this VR system for applications such as: medical training and medical visualization, architectural planning, and mainly for scientific visualization, see Figure 2.16.

In the classical Workbench concept only one person is being head-tracked for setting up the stereo perspective. Other persons by the Workbench get a slightly distorted view. This problem has been addressed with the Two-user Responsive Workbench, which supports individual stereoscopic views of the two users and enables collaboration in a shared space [Agrawala *et al.*, 1997].

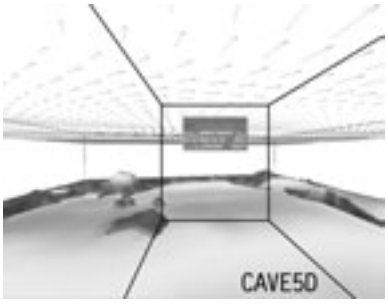
Certain applications can profit from the life-size display in the fully immersive CAVE-like systems. As the CAVE provides a much higher degree of user immersion in the virtual worlds, applications like architectural walkthroughs, or navigation through large computer generated environments are very impressive. Also a large number of applications from the field of scientific visualization have been tested within this environment. Indeed, scaling of the 3D visualizations of data from desktop into the 27 cubic meters of space is impressive in itself. Yet a not clearly answered question is whether the scientific data really need full human-size immersion of the user. It seems that Workbench systems or desktop personal VR systems are good enough, and even better for many scientific or clinical applications which should be employed in daily use. However, CAVEs are imposing systems. Due to the (almost) completely surrounding projection these systems offer better immersion in a larger environment, but it is much harder to find scientific applications that would really need a CAVE.

Cave5D and Cave6D systems (see Figure 2.17) are examples of quite popular visualization programs for the CAVE. They have been adapted from Vis5D, a desktop atmospheric data visualization program, for the CAVE. Vis5D [Hibbard *et al.*, 1996; Hibbard & Santek, 1990] is a visualization library that provides techniques to visualize multi-dimensional numerical data from atmospheric, oceanographic, and other similar models, including iso-surfaces, contour slices, volume visualization, and wind vectors. The Cave5D framework [Wheless *et al.*, 1998] integrates the CAVE library with the Vis5D library in order to interactively visualize atmospheric data in the Vis5D file format in the CAVE or on the ImmersaDesk. A collaborative environment has been incorporated in Cave6D [Kapoor *et al.*, 2000]. Each participant runs a separate instance of Cave6D. The tele-immersed participants see the virtual environment from their own perspective, and can freely navigate in the space. The immersed participants share the same virtual space.

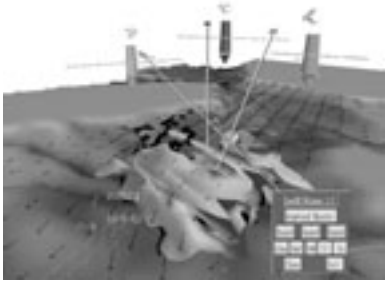
A system called TIDE (Tele-Immersive Data Explorer) [Sawant *et al.*, 2000] has demonstrated distributed interactive exploration and visualization of large scientific datasets. It uses centralized collaboration and data-storage model with multiple processes designed to allow researchers around the world to collaborate in one shared and interactive VE.



(a) Vis5D



(b) Cave5D



(c) Cave6D

Figure 2.17: Visualization of atmospheric data on desktop (Vis5D), in an immersive VE (Cave5D), and in a collaborative VE (Cave6D). [Image source: Hibbard, Wheless, Kapoor (EVL)]

Figure 2.18: Studier Stube visualization environment is based on see-through HMDs and electro-magnetic tracking. The system is operated with the Personal Interaction Panel. [Image source: Schmalstieg et al., VR-VIS Center (Austria)]

A different approach to immersive visualization is presented by the "Studierstube" approach [Schmalstieg *et al.*, 1998]. The designers present a client-server architecture for multi-user collaborative visualization, presentation and education. Studierstube is based on augmented reality technology and see-though HMDs. These displays do not affect natural communication and interaction of collaborating users, see Figure 2.18. Each of the users has his own view on the model and can independently use different layers of data for visualization. This system uses the so-called Personal Interaction Panel (PIP) to assist user interaction and to control the application. Due to recent great improvements of HMDs and see-through HMDs, this augmented reality approach shows a good alternative for immersive visualization instead of the large projection systems.

Figure 2.19: Dextroscope system and VIVIAN medical visualization software [Image source: L. Serra, KRDL (Singapore)]

As mentioned earlier, clinical application of VR technology must consider ergonomic and economic aspects. The current HMDs or the large projection systems can hardly offer a "user friendly" working and affordable environment. It is a severe shortcoming of these technologies. Let's consider for example medi-

cal application of VR (surgery training and planning). Nobody would ever think about using CAVE, Workbench or HMDs the whole working day in this case.

A good example towards ergonomic VR for a clinical use is the Dextroscope, a desktop-size Virtual Workbench, see Figure 2.19. This approach and its application in medical visualization has been presented by Serra et *al.* [1997,1999]. They have also developed the VIVIAN system for volumetric tumor neurosurgery planning. It supports volume rendering and data segmentation to outline tumors, and the trajectory planning for surgery.

It uses 2D/3D paradigm of interaction, which combines 3D direct manipulation of volumetric data with unambiguous 2D widget interaction. The 2D interaction is achieved by providing passive haptic feedback of the virtual control panel with the widgets whose position coincides with the physical table.

2.3.2 Visualization in VR: Research Issues

Besides the already mentioned general research issues in Virtual Reality, we can extend the general research agenda for scientific visualization in immersive VR by the following issues:

- Interaction and visualization techniques that are scalable with data and model size
- Feature extraction and tracking techniques for large time-dependent data
- Time-critical computing and visualization techniques for VR
- Tele-collaboration in immersive visualization environments
- Effective computational steering and visualization tools and environments

Unfortunately, most of the interaction and visualization techniques, developed so far, are based on small-scale problems. Techniques that work well with 100 objects in a scene will not necessarily work with 10.000 objects. For example object selection: one from 100 or from 10.000. Obviously, it is similar for visualization techniques. For example iso-surface extraction in a dataset with dimensions 64x64x64 may be still performed interactively, but 256x256x256 already becomes a problem. Time-critical computing and visualization techniques can offer a solution [Bryson & Johan, 1996; Funkhouser & Séquin, 1993].

Let's consider a large time-dependent dataset with thousands of time-steps. Classical data visualization techniques will probably fail. Part of the solution may be in the use of interactive exploration in VR combined with feature extraction and tracking techniques [Reinders, 2001; Sadarjoen, 1999].

As scientific problems are often multi-disciplinary, we need to encourage collaboration and co-operation of experts. VR can provide good environments also for collaborative visualization.

Computational processes and simulations need more effective steering and visualization. VR also has a great potential in this area.

For further reading on the hot topics of VR and research issues in immersive visualization we refer to recent reports [van Dam *et al.*, 2000; Brooks, 1999].

2.4 Research Agenda of This Thesis

Many years ago R. Hamming stated that *"the purpose of computing is insight, not numbers [Hamming, 1962]."* The most important aspect of this research was to prove the concept that visualization of scientific data in immersive Virtual Reality significantly improves the process of data exploration and discovery, and that VR helps to get better oriented in the multi-dimensional datasets and contributes to gaining scientific insight.

We have learned that a good way to get insight from numbers is to visualize them [McCormick, 1987]. As the size and complexity of computational simulations grows, the amount of data will also increase. Therefore, new concepts of intensified interactive visualization should be developed. One such concept is the use of VR for interactive visualization and exploration of scientific data.

Based on the general research issues of Virtual Reality (see Section 2.2.5) and visualization in VR (see Section 2.3.2), we have proposed the following research agenda for this thesis:

- Study of existing visualization systems for possible VR extensions
- Development of an experimental visualization environment for the Virtual Workbench
- Study of existing VE interaction techniques and development of new interaction techniques for more effective object selection and manipulation, and for navigation in and exploration of VEs
- Study of existing and designing alternative force feedback approaches
- Incorporating simulation into the visualization process and implementation of a computational steering environment
- Design and implementation of interactive exploration and visualization tools for VR
- Proof of the concept by the case studies, collaborating with other research groups, and trying to solve their scientific questions with VR
- Trying to define the essential contribution of VR to data visualization

All these topics will be discussed in more detail in the following chapters.

Chapter 3

The Concept of the Virtual Workbench

This chapter begins with an introduction to the Virtual Workbench (Section 3.1). Followed by a description of technical aspects of this system (Section 3.2), such as the hardware configuration, tracking system and input devices, and projection and viewing of virtual worlds. Design aspects of VEs for data visualization, including requirements on the user interface and interaction, are discussed in Section 3.3. The software environment, represented by the RWB Library and the RWB Simulator, is described in detail in Sections 3.4 and 3.5

3.1 Introduction to the Virtual Workbench

The Virtual Workbench is a Virtual Reality system for 3D visualization and 3D interaction. In this thesis we will call this system the Responsive Workbench (RWB) or equivalently the Virtual Workbench. The original RWB concept was developed by [Krüger *et al.*, 1995].

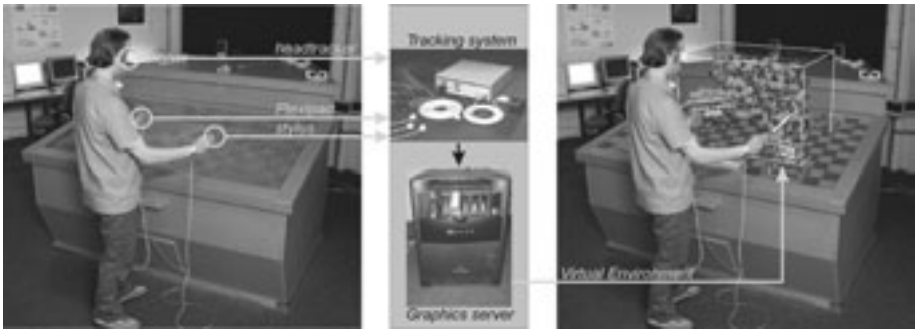


Figure 3.1: Overview of the Virtual Workbench. The left image is a photo of a VR session. In the right image, the 3D world that the user experiences is superimposed on the original photo.

The RWB provides a semi-immersive virtual environment, see Figure 3.1. The user stands by a table with the upper half of the body immersed in a VE, using the physical table for reference and support. In this approach the user is not fully immersed and is always aware of the surrounding natural environment. The user can look away from the VE and see the surrounding real world such as his body, the table, the physical tools and props being used and other people in the room. This is in contrast to many other Virtual Reality systems

such as CAVE or HMD, which offer fully-immersive environments. Instead of bringing the user inside the virtual world, the user at the workbench looks down on virtual objects that are brought into the real world [van de Pol *et al.*, 1999].

The RWB is in some sense complementary to the CAVE [Cruz-Neira *et al.*, 1993]. However, the usage of the RWB is different from the CAVE. The RWB benefits from the table metaphor although its field of view is rather limited. In the CAVE, all objects are usually virtual. In automotive industry, are put mock-ups, car-seats inside the CAVE to have at least something real with a substance to be able naturally interact with the virtual environment. In the CAVE usually larger objects are visualized, and the user interacts with them often at a larger distance than on the RWB. However, the Virtual Workbench is for using of our applications more suitable than the CAVE. Nevertheless, the CAVE excels at demonstrations in the final stage of visualization projects, see Section 6.2.7.

3.2 Technical Characteristics of the Workbench

The Virtual Workbench is a large table with a tilted, frosted glass surface which serves as a projection screen. This Workbench was manufactured by TAN systems [Web-TAN]. A CRT-projector mounted inside the wooden framework is used to project images via a series of mirrors on this glass surface (Figure 3.2). To obtain a clear and sharp image the CRT projector has to be well calibrated and precisely aligned. In addition to this back-projection system, LCD shutter glasses, a tracking system and a graphics server provide the VE. The user standing in front of the RWB, wearing head-tracked shutter glasses, can experience the computer generated VE and interact with virtual objects on the Workbench.

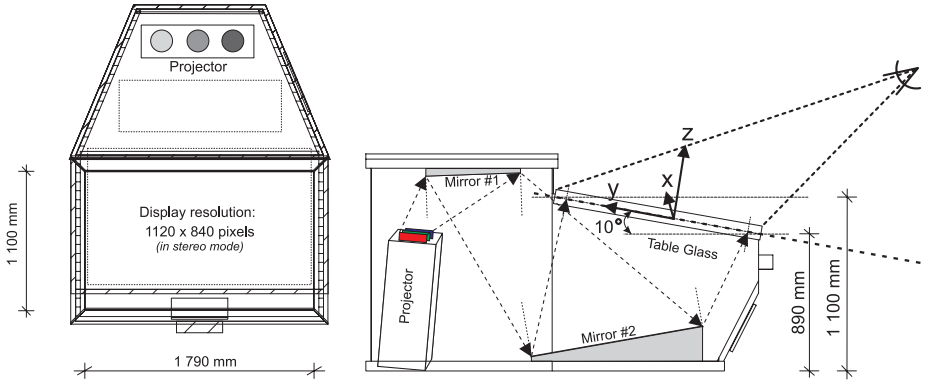


Figure 3.2: Schematic overview of the Workbench: top and side views

The heart of the VR system is the SGI Onyx2 graphics server [Web-SGI-Onyx], which generates the stereo images for the Workbench and handles the user input. The graphics server is equipped with four 195 MHz 64-bit MIPS R10000 processors, 1.5 GB of shared memory, and an InfiniteReality2 graphics

card with 64 MB of texture memory. This server is designed for high performance graphics applications. The hardware architecture enables an optimal usage of multi-processing and one or more graphics boards. The stereo effect on the Workbench is achieved by using active stereo projection. We use a display resolution of 1120×840 pixels. Two images for the left and right eye are alternately projected on the Workbench at a frequency of 96 Hz (=2x48 Hz). At the same frequency, the StereoGraphics CrystalEyes LCD shutter glasses [Web-Stereographics] block the light falling on one of the eyes.

3.2.1 Tracking System and Input Devices

The tracking system enables dynamic spatial measurements of the tracking sensors. The RWB is equipped with a Polhemus Fastrak [Web-Fastrak] system, consisting of a control unit, an electromagnetic field emitter and one or more tracking sensors. The control unit is connected via a serial port to the Onyx2. The system is able to determine both spatial position (XYZ) and orientation (AER: azimuth, elevation, roll) of each of the sensors. This results in a total of six degrees of freedom (6 DOF) per sensor. For a later use we must convert orientations to the angle notation of Iris/OpenGL Performer (HPR: head, pitch, roll).

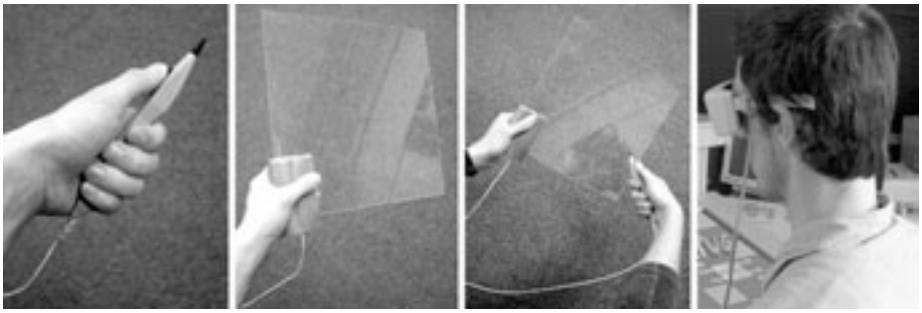


Figure 3.3: The stylus, the Plexipad and their two-handed use (left). The head-tracker is mounted on a pair of shutter glasses (right)

In our configuration three tracking sensors are used as follows. One tracker sensor is mounted on a pair of stereo glasses and is used to measure the position and orientation of the user's head *(head tracking)*. As described earlier, this viewpoint tracking simulates the motion parallax depth cue. The other sensors are used for two input devices which can be used simultaneously: the stylus and the Plexipad, see Figure 3.3.

- **Stylus:** a pen-shaped input device with a single button. The pen can define a point in space. The shape of the pen defines a directional reference axis. The pen can be used to specify a line in 3D space. This function is often used for a ray-casting selection.

- **Plexipad:** a lightweight transparent acrylic panel (300 x 300 x 2 mm) with a tracker sensor mounted under a foam handle at the edge of the panel. The handle allows a firm and comfortable palm grip, reducing fatigue in the fingers. With the tracker mounted close to the wrist, the inconvenience of the tracker cable (obstruction of the view, weight on the panel) is reduced to a minimum. The panel can be used to position and orient a 2D plane. It defines a 2D reference plane in 3D space.

These input devices can be used in two-handed interaction scenarios, using both hands simultaneously in a single task. The issues of one-handed and two-handed interaction are discussed in detail in Section 5.1.

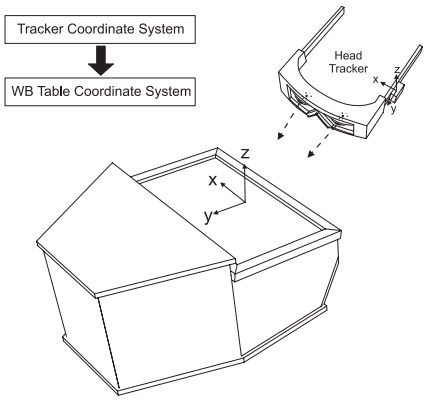


Figure 3.4: Tracker to table coordinate transformation

The *tracker daemon* is a separate process, which runs on the SGI Onyx2. It periodically reads (50Hz) the data from the tracking system, converts them to the workbench table coordinate system and stores them in shared memory. The tracker daemon also offers functions to access and read the data for any running process.

The view update function of the RWB Library (see Section 3.4) reads the new position and orientation from the tracker daemon's shared memory. This information is first stored in the *tracker coordinate system* in a form of 4x4 matrix. The position is placed in the translational part of the matrix and the HPR rotation is stored in the rotational 3x3 sub-matrix (Euler rotational matrix). This matrix forms the coordinate frame. Originally, this frame is defined with respect to the tracker coordinate system, thus it has to be transformed to the *workbench table coordinate system*, see Figure 3.4. Therefore, a 4x4 tracker to table transformation matrix is used. The original head-tracker coordinate frame is transformed by this matrix. The resulting frame has to be aligned so that in the neutral orientation of the head tracker, resp. stylus, its Z-direction is pointing upwards in the workbench table coordinates. This is done by multiplying the frame with pre- and post- rotational matrices. The final frame together with the information on

the offsets of the eyes is used to position the viewpoints of the left and the right eyes. The viewing direction is always pointing towards the center of the RWB table, see Figures 3.4 and 3.5.

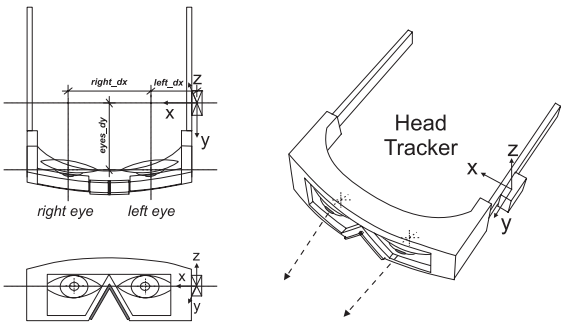


Figure 3.5: Head tracking and eye positions

3.2.2 Registration and Calibration of the Tracking System

Proper registration and calibration of the tracking system is very important, see Figure 3.6. *Registration of the tracking system* means that we define a transformation from the original tracker coordinate system to a known world coordinate system. *Calibration of the tracking system* should minimize the tracking errors. Registration of the table coordinate system is done by defining the tracker to table transformation. The user clicks with the stylus on 3 points $\oplus$: the origin B and points on the X and Y axis of which we know the exact position on the screen (in pixels) and their exact position with respect to the center of the glass-plane (in cm), see Figure 3.6. We measure the positions in tracker coordinates. From this we can obtain the scaling factors, the axis-vectors X, Y and the Z-axis, which is a cross product of X and Y vectors, and the position of the origin with respect to the tracker coordinate system. The X, Y and Z axis-vectors must be normalized. From the three axis-vectors and origin B the orthonormal frame base (4x4 matrix) is constructed and its inverse is the tracker to table transformation matrix M_{TT}:

$$M_{TT} = \left[\begin{pmatrix} X_i & X_j & X_k & 0 \\ Y_i & Y_j & Y_k & 0 \\ Z_i & Z_j & Z_k & 0 \\ B_i & B_j & B_k & 1 \end{pmatrix} \right]^{-1} \star M_S$$

The matrix M_S converts screen pixel metrics into the metrics of the virtual world. M_{TT} matrix is used to convert tracking sensor matrix M_T from tracker coordinates to the virtual world coordinates M_W aligned with the table.

$$M_W = M_T \star M_{TT}$$

After a proper tracker registration we can measure positions of the sensors relatively accurate close to the tracking emitter, which is mounted in the center of the front wall of the Workbench, see Figure 3.6. However, when the tracking sensors are positioned more than 50 cm away from the emitter, as frequently happens, then the tracking errors in position measurement can reach even 2 or 3 cm. To compensate the tracking errors we designed a grid calibration scheme. The grid calibration is performed by measuring the tracking error on the grid. During this calibration procedure the user has to click as accurately as possible on the grid markers. We use a bi-linear interpolation scheme to correct for these errors.

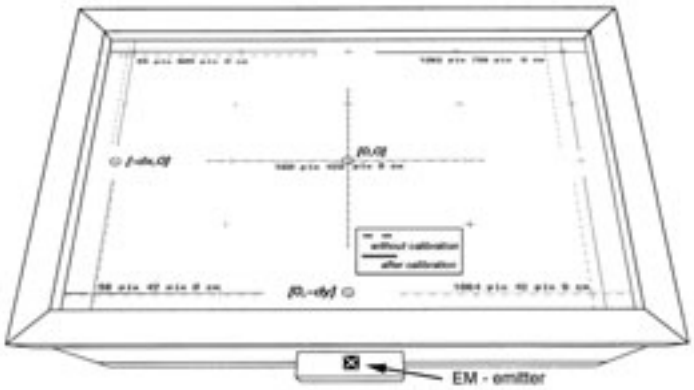


Figure 3.6: Tracker registration and grid calibration of the tracker data

The results of the calibration procedure are stored in a tracker calibration data file, which is later used by the tracker daemon for correction of the tracking data of the sensors (position only).

In Figure 3.6, the dashed lines show stylus tracking results in the plane without the grid calibration. The solid lines show the calibrated results. The tracking errors are very annoying especially in a direct 3D interaction with the stylus when the virtual cursor is sometimes noticeably displaced from the stylus position. Similarly, the tracking errors occur also when measuring the orientation of the sensors. For the head-tracking on the RWB the tracking errors, both in position and in rotation, are less significant and the view distortion is minimal.

For the stylus the error in rotation can only be observed when using ray-casting selection (laser pointer metaphor). We don't see it as a big problem because the user more is interested about the place of the ray intersection with virtual objects. From experience we know that users can easily compensate for a slightly bent ray because they get the visual feedback of the ray and it smoothly follows the motion of the stylus. Therefore we did not implement any general scheme for angular calibration.

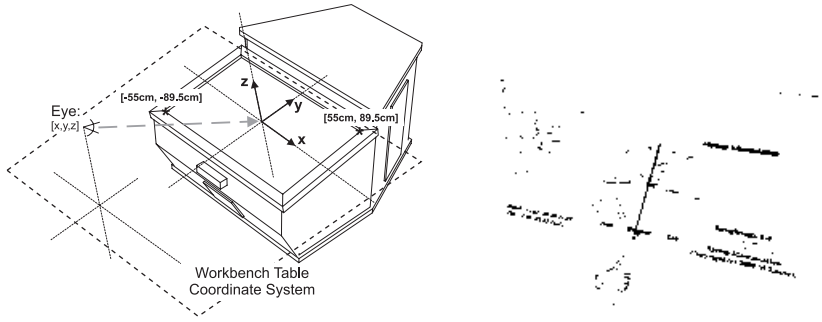
In the case of the Plexipad the angular tracking errors will cause a slight misalignment between the transparent pad and virtual objects that are placed on

its surface. A future improvement of the tracking system would be an improved dynamic calibration scheme, which can also cope with angular tracking errors [Kindratenko & Bennett, 2000].

In addition to the coordinate transformations and calibration of the tracking data, the tracker daemon also integrates actual velocity and acceleration vectors of the stylus. We use a simple low-pass filter to minimize the noise in the tracker data. The computed velocity and acceleration of the tracking sensors can be used by any RWB application for recognizing gestures. For example, if an application needs to detect the instant when the stylus stops, it can easily check the actual velocity of the stylus.

3.2.3 Projection and Viewing

The virtual world projected on the RWB screen is represented in the workbench table coordinates. The tracked positions and orientations of the user's head and the stylus are in the tracker coordinate system and therefore must be converted into the workbench table coordinate system. We use a table-centered and table-aligned coordinate system, see Figure 3.7(a). In the RWB environment, the head-tracker updates user's viewpoint, and the tracking of the stylus forms the base for 3D interaction, see Figure 3.7(b).



(a) (b)

Figure 3.7: (a) Workbench table coordinate system (b) 3D user interaction

In common 3D rendering systems the user's eye is assumed to be positioned on the axis of perspective projection, so called *on-axis-perspective.* The viewpoint is on the positive Z-axis and the viewing frustum is oriented into negative Z-axis. For this perspective projection to be correct, the user's head should be on the central axis of the screen. For monoscopic images such a viewpoint assumption is not a problem, but for stereo images the movements of users's head cause a distortion of the perspective.

On the Workbench we cannot assume that the user's eyes are on the Z-axis. Therefore we have to use an *off-axis-perspective.* The construction of the RWB perspective frustum is shown in Figure 3.8.

We have to set up the perspective frustum from the user's eye position (in table coordinates) pointing down, perpendicular to the workbench ground plane. After the perspective transformation we have to perform a 2D shift in viewport coordinates to fixate the viewport origin with the origin of the RWB. This is equivalent to the assumption that the user is always looking at the center of the Workbench. By this we have fixed the ground plane of the projected VR world to the screen of the Workbench. This has to be done for the left and the right eye to set the correct stereo perspective.

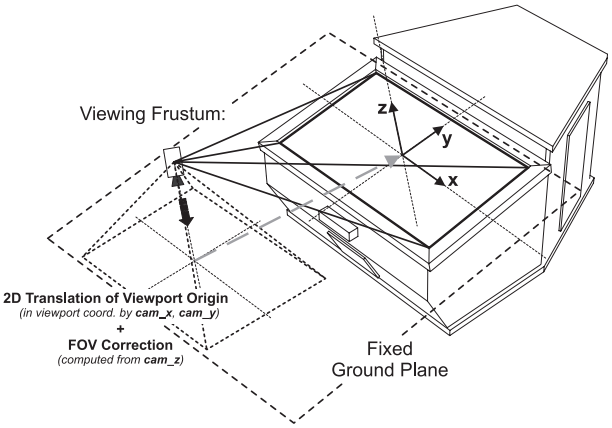


Figure 3.8: Construction of the viewing frustum

The following code fragment shows the procedure of setting the view channel and the off-axis perspective frustum in the RWB Library. A basic overview of Iris Performer [Eckel *et al.*, 1997], a high performance rendering and scene graph toolkit, which we use in the RWB Library, is given in Section 3.4.2.

It is important to note that Iris Performer uses a different notation of viewing direction than usual. Performer's viewing direction is the positive Y-axis, and OpenGL uses the negative Z-axis.

```
EyeViewMatrix.makeEuler(0.f,-90.f,0.f);  //*** rotation matrix from Euler angles(H,P,R)
                                         //*   Viewing direction: Performer +Y, OpenGL -Z
EyeViewMatrix.setRow( 3, eyeTable   );   //*** insert the eye point (in table coordinates)
                                         //*   into the viewmatrix

float projFactor = zNear / eyeTable[2];  //*** compute window boundaries in zNear plane
WinLeft   = (tableMin[0] - eyeTable[0]) * projFactor;
WinRight  = (tableMax[0] - eyeTable[0]) * projFactor;
WinBottom = (tableMin[1] - eyeTable[1]) * projFactor;
WinTop    = (tableMax[1] - eyeTable[1]) * projFactor;

tableAspect = (tableMax[0] - tableMin[0]) / (tableMax[1] - tableMin[1]);
//** tableAspect =  (89,5 - (-89,5) = 179 cm ) / (55 - (-55) = 110 cm )

//*** setup of the viewing frustum and perspective projection via Performer's pfChannel
ViewChannel->setViewMat(EyeViewMatrix);
ViewChannel->makePersp(winLeft, winRight, winBottom, winTop);
ViewChannel->setAspect(PFFRUST_CALC_VERT, tableAspect);
ViewChannel->setNearFar(zNear, zFar);  //*** zNear = 0.01 cm;   zFar = 10000.0 cm;
```

3.3 Design Aspects of the VE on the Workbench

The user interface with its interaction methods plays a crucial role in any application. The user interface has to expose the application's functionality, allowing the user to perform his tasks and effectively control the behavior of the application. A virtual environment provides the user interface for VR applications. By using spatial input devices, the user can interact with this computer-generated 3D environment. In the design of the user interface of our system we concentrated on the user's tasks in scientific visualization and the specific characteristics of VR on the RWB. These aspects were taken in account in the design of the visualization tools, the interaction scenario's and the VE itself.

3.3.1 Visualization tasks in the VE

The typical processing steps of the scientific visualization pipeline have been described in Chapter 2. We can distinguish two separate stages in a visualization application: *development and execution of the visualization pipeline.* Network application builders (AVS, OpenDX, etc.) or visualization libraries (VTK) are used in the development stage. In our approach, visualization applications for the Workbench are programmed in C++ using the VRX toolkit and the RWB Library and Simulator. The design and development of the applications can be more effectively performed in a conventional desktop computer environment than in a VE. However, the most obvious advantages of VR can be expected in the execution stage. In order to effectively support the visualization in VR, the layout of virtual objects in the VE should be done carefully with respect to the capabilities and constraints of the RWB. The user on the Workbench has an active role (not passively watching 3D animations) and has to be provided with a set of interactive tools.

The main user task is to control the different stages of the visualization pipeline via interaction (selection and manipulation) with individual objects of the VE. We distinguish the following interaction tasks: *navigation* (positioning, orienting, selection, cropping and zooming of the data), *visualization mapping* (choice of visualization method and setting of parameters; i.e iso-surfaces, color maps, transfer function, seed points), *probing and data exploration* (visualization and virtual measurement) and *computational steering*. We have implemented support for these tasks by allowing direct spatial interaction with the data using a set of virtual exploration tools; for a detailed description, see Chapter 5.

Although a large part of the interaction with the visualization application can be performed using spatial control, some abstract tasks, such as setting a value or enabling or disabling a function, do not have an intuitive spatial representation or a straightforward real-world metaphor. Therefore, we use virtual objects or interaction widgets for this abstract control from the VE. In our implementation, we used simple interaction widgets such as buttons (switches) and value sliders, analogous to the well known graphical user interface elements in desktop applications.

3.3.2 Workbench Viewing Metaphors

As described in the previous sections, the RWB brings virtual objects in the real world. In contrast to fully-immersive VR environments, the user is not surrounded by the VE but looks down onto it. This and other VR and RWB specifics should be taken into account when developing an application for the Workbench. The Workbench concept offers two basic metaphors: *the laboratory table metaphor* and *the window-on-the-world (WOW) metaphor.*

Using the laboratory table metaphor, the user experiences the VE as if the virtual objects were resting on top of the table. This scheme is an excellent analogy to working on real experiments at a laboratory table. We draw a background image exactly aligned with the frosted glass plane to emphasize the physical table surface. When working with large volumes of data, a serious constraint is the limited height of the VE above the table. Because the user's view frustum is limited to the screen size, virtual objects positioned just outside this field are culled (clipped), seriously disturbing the perceived stereo effect, see Figure 3.9.

This limitation can be avoided by restricting the height of the VE or by using the WOW metaphor. In this scheme the user experiences the tabletop glass as a window onto a VE lying mainly under the table, allowing a birds-eye view on large worlds. However, the WOW metaphor has also several limitations, such as: conflicting cues with real world and no natural reference plane; when objects are out of reach, remote selection and manipulation is needed.

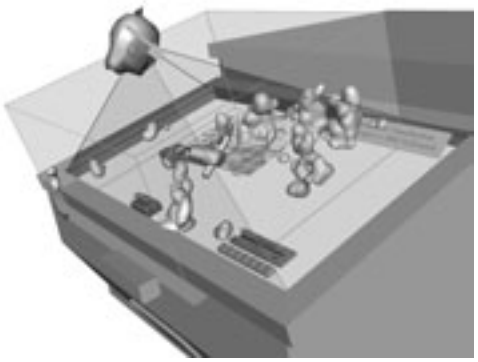


Figure 3.9: The field of view is limited by the screen size. The view frustum indicates which part of the VE and the data space remains correctly visible.

The physical shape of the RWB also has its implications on the types of interaction. Logically, the input devices, which are used to interact with the VE, can only be positioned above the glass surface of the table. If the tabletop metaphor is used, the projected VE and the real space above the table coincide: the virtual space and real space are aligned. This allows users to directly interact with virtual objects. This direct interaction with the VE is done intuitively at the exact location of the input devices used. A typical example is the use of the stylus

to select and manipulate an object directly in the VE by positioning the stylus inside the virtual object, see Figure 3.10 (left).

This is in contrast to remote interaction where the input devices are used to interact on a remote location in the VE. An example of remote interaction is ray-cast manipulation, where the stylus is used to cast a ray on remote objects in order to select and manipulate them. If the WOW metaphor is used, remote interaction is the only type of interaction available as the user cannot move the input devices under the glass surface, see Figure 3.10 (middle). However, many users prefer the direct interaction because the remote interaction is less accurate and less intuitive.

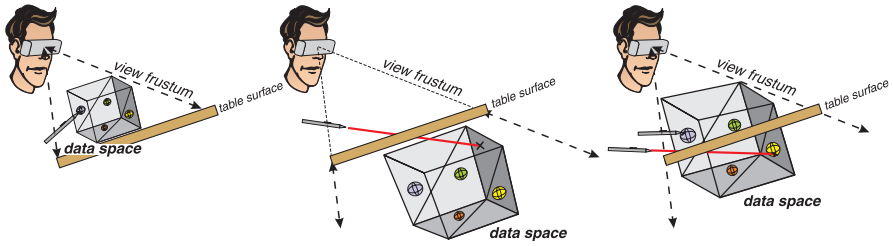


Figure 3.10: Viewing metaphors: table metaphor and direct interaction (left), window-on-the-world (WOW) metaphor and remote interaction (middle) and combined metaphor, using both direct and remote interaction (right)

Our solution is to combine the advantages of both metaphors, providing both an overview on data as well as intuitive direct interaction. We place the VE partly above the table and partly underneath, instead of constraining the VE to be either above or underneath the table surface. By using ray-cast manipulation, remote virtual objects can be repositioned into the reach of the user to allow direct manipulation, see Figure 3.10 (right). For example, we have defined a virtual data space object, which can contain both volumetric and object data. The data space is represented by a box floating in the VE, using an outline to indicate its spatial boundaries. Interesting parts of virtual objects (such as the data space) can be brought above the table for detailed inspection while keeping an overview on the object as a whole. The user can operate the visualization application by directly interacting with this data space object alone or in combination with virtual exploration tools. When only a wire-frame bounding box is used for the indication of the data space, it cannot be selected by ray casting. To enable the ray-cast selection, we have added a transparent bounding box constructed with inward facing triangles. Due to the back-face culling the front sides of the box will not be rendered and will not stay in the way, so that the back sides of the box (inward facing) could be intersected with the ray.

Although we do not explicitly emphasize the table surface of the display, the user should be aware of the table surface intersecting the VE (data space). It physically divides the VE into direct interaction and remote interaction parts.

3.3.3 Multi Sensory Feedback

Without appropriate feedback it is difficult to perform any type of interaction. By providing feedback the user is informed about the progress or results of performed actions. In VEs we can usually employ visual, acoustic, tactile and force feedback schemes. Naturally, the primary feedback of actions is given in every frame by the graphical update of interaction cursors and manipulated virtual objects. However, freely selecting a point in 3D space or even selecting a virtual object can be a difficult task without additional feedback.

This feedback can be given when an event has been detected, such as if an object has been selected, a button has been clicked or if the ray has been activated. By using a visual (graphical) feedback, e.g. highlighting an object when selected, the user can be informed effectively. For example, in our case the currently selected object or interaction widget is colored red. Further, we use simple sound effects to accompany actions to enhance the pure visual feedback.

A more physical feedback is tactile or passive haptic/force feedback. The glass surface of the Workbench can be used to support the interaction and create additional physical constraints. When widgets in the VE are correctly positioned on top of the table surface, the tactile feedback can enhance the control of these tools. Instead of freely positioning the stylus in space to select a widget, the user only has to touch the glass with the stylus. This tactile feedback can also be provided by the acrylic surface of the Plexipad, see Figure 3.11. By combining the regular visual feedback with both tactile and acoustic feedback the interaction can be improved.

Figure 3.11: A user touches the surface of the Plexipad to select a button, and is simultaneously provided with visual, acoustic and tactile feedback.

As our hardware configuration of the Virtual Workbench does not contain any active force feedback (haptics) device (such as a Phantom [Massie & Salisbury, 1994]), we cannot provide active force feedback during manipulation of objects in the VE. Therefore, we have developed visual force feedback tools, see Section 4.2.

3.3.4 Layout of the VE

The user's position with respect to the RWB also has its implications on the layout of the VE. The users prefer direct interaction to remote interaction. Thus, most of the interaction should rather take place within direct reach of the user. In addition, the tracking system is most accurate close to the electro-magnetic emitter, which is mounted at the front center of the RWB. In most cases the user stands at a centered position before the emitter, with the stylus in the right hand (assuming a right-handed user). To improve interaction speed and to reduce fatigue, the most direct interaction should take place within direct reach of the user's dominant hand. The dominant hand holds the stylus, and is the preferred hand for precise movements such as writing. For most people this is the right hand, but it is easily adaptable for the left-handed users.

The ideal position of the data space is in front of the user slightly above the table to allow direct interaction. Frequently used widgets such as buttons and sliders should therefore be placed as close to the user as possible, without blocking the view on the VE (Figure 3.12).

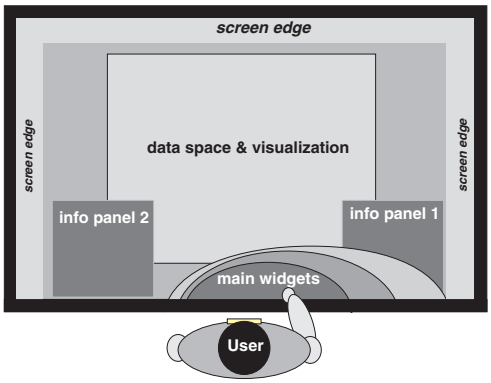


Figure 3.12: Preferred layout of regions for interaction, display of information, and data visualization on the RWB

In a two-handed configuration the Plexipad can be also used to improve usability. The Plexipad can provide a passive tactile/haptic feedback and form a container (background) for interaction widgets. The two-handed interaction aspects are discussed in more detail in the Section 5.1.

3.3.5 Technical Constraints

The RWB is a back-projection system, and as such has its advantages and disadvantages. An advantage, compared to front-projection systems, is that we do not have problems with shadows, caused by a user standing between the projector and the projection screen. A disadvantage of the back-projection approach is that the user's hands with interaction devices block/hide a part of

the VE. This problem can be minimized by a careful design of the VE, suitable interaction techniques, and a convenient placement of interaction cursors (not behind/under the hand with the stylus).

Small Objects and Fine Details

Although the screen of the RWB provides a large display space, not every region can be used with the same effectiveness. A drawback of the large projection screen is that the limited sharpness and focus of the images disturbs small details (several pixels in size). It is especially noticeable when using small detailed objects or text far away from the user.

Another problem with very small objects is that they are difficult to select due to the tracker inaccuracy. Thus, small detailed objects, which represent important information, should be positioned close to the user or should be avoided. Naturally, enlarging of objects should not be exaggerated. Large objects or control widgets can obscure the view on the data.

Use of Colors in the VE

In most visualization applications, a careful use of color is essential for the visualization of information. In VEs, we also have to work very carefully with colors. Most of the projection systems, including the RWB, show colors differently than CRT monitors. They often lack in contrast, brightness and sharpness and have a distorted color spectrum. This even becomes worse, if the VR lab is not entirely darkened. Also the shutter glasses or polarized glasses somewhat weaken the color spectrum.

In VEs the colors are often manipulated to provide essential depth cues. Techniques like lighting, shading, shadows, and fog-haze can provide essential spatial information of virtual objects in the VE. In our case study of Molecular Dynamics, the depth fog-haze was especially useful. When the view was cluttered with particles of the same color, the depth fog-haze provided an additional depth cue in distinguishing nearby and distant particles, see Section 4.3 and also the Color Section.

From our experience we know, that parts of the VE such as textual information and user interface widgets should be bright and contrasting, and therefore clearly visible.

Performance Implications

To achieve a convincing visualization experience in VR, we have to provide interactive and responsive environment [Krüger *et al.*, 1995]. On the one hand, this means that we have to represent the data using the most effective and detailed graphical representation, using correct depth cues to enhance the visual information. On the other hand, the system must provide the highest possible scene

rendering rate and interactive user feedback. Considering that the performance of the hardware and software has its limits, we have to find a balance between detail and speed.

For an interactive VR application the absolute minimum for the scene update frequency is ten frames per second (10 Hz). Below this frequency the VR experience is seriously disturbed. A slow update rate causes the latency to grow. The end-to-end system latency is the time spent between the user actions and the resulting system feedback. Especially when the head tracking is used to update the perspective view on the scene, the system latency can cause nausea or simulator sickness. More about performance aspects of RWB Library can be found in Section 3.4.4.

3.4 RWB Library: A Software Environment for the Virtual Workbench

3.4.1 Introduction to VR Software for the Workbench

The VR software infrastructure begins with software interfaces to VR devices. The VR hardware, such as VR displays, tracking systems and 3D interaction devices, is not usually delivered with all the necessary software. Usually it comes with software drivers for various platforms only. Some commercial or publicly available VR software has support for a variety of VR hardware. However, it is not easy to choose for the right VR software. Either it is too general or too much application specific.

Existing VR Libraries

In summer 1998, when the RWB facility was installed at the High Performance Applied Computing Center (HPαC) at TU Delft, there were not many software options. The Virtual Workbench was intended to serve as a high performance visualization system. It should work in a cluster with the other supercomputers such as CRAY T3, SGI Origin 2000, or Beowulf (Linux) Cluster.

The intention was to maximally utilize the resources of the SGI Onyx2 and to create an open VR data visualization platform for the Virtual Workbench. Another aspect was to implement a computational steering environment, which would enable the user to control a supercomputer simulation directly from a virtual environment displayed on the Workbench.

In the next two years we have studied various VR and visualization systems. Our visualization group at that time (until 1999) has made mainly use of commercial data visualization systems such as AVS 5 [Upson *et al.*, 1989], AVS Express and Iris Explorer. Our first idea was to investigate the use of existing VR interfaces with commercial visualization systems. Such solutions were the RWB-viewer for Iris Explorer [Smychliaev, 1998], the CAVE-lib viewer for AVS 5 [Web-AVS-int] or the Grotto-viewer for AVS 5 [Web-VR-lib], see Figure 3.13.

Figure 3.13: Grotto-viewer for AVS 5.0: a visualization network of AVS on the computer screen (left), a view on the Workbench (right)

These VR extensions were visualization network modules providing a VR view on the data with very limited interaction and navigation. We concluded very soon that visualization in VR has to offer much more than passive exploration of visualization results without having possibility to adjust visualization parameters, such as changing the iso-value in case of iso-surfaces. Another aspect was that AVS and Iris Explorer were quite "closed-source" systems and external developers did not have an access to the source code of the modules. For example, how the Geometry Viewer module for AVS 5 works will remain unknown and maybe therefore there have never been any "good" VR viewers for AVS that supported 3D interaction with the visualization network. In AVS it was not easy to implement a connection between a 3D widget and the visualization network. For example, a slider in the virtual environment, which could control an iso-value of an iso-surface module of the visualization network.

At the time of writing this thesis (2002), for example an AVS/Express Multipipe Edition is available [Web-AVS-int] that provides a better interface to VR systems; however, the interaction with visualization directly from the VE network remains primitive. Systems such as Open DX [Lucas *et al.*, 1992] and VTK [Schroeder *et al.*, 1999] are open-source and well documented. It gives us the opportunity to develop better environments for visualization.

In addition to various VR-viewers for visualization systems, there are many experimental VR libraries, such as for example Avango [Dai *et al.*, 1997], VR-Lib [Web-VR-lib], MR Toolkit [Web-MR-toolkit], VrTool [Web-VR-toolkit], SVE-lib [Web-SVE-lib], Studierstube [Schmalstieg *et al.*, 1998], Aura & VIRPI [Germans *et al.*, 2001], CAVE5D [Web-CAVE5D], COVISE [Rantzau *et al.*, 1998], used by VR researchers for various purposes. Some of these libraries are freely available.

VR libraries, such as VR-Juggler [Bierbaum *et al.*, 2001], MAVERIK [Hubbold *et al.*, 1999] follow the global trend and are available as open-source software. This gives us an opportunity to work together on improvements. Other software libraries were commercially available, such as the CAVE Library [Web-CAVElib] or the WorldToolKit (WTK).

We have tried several of these libraries but none of them has fulfilled our expectations. Today we can see a variety of VR systems and libraries more or less specialized on specific types of VR applications, for example data visualization, industrial or medical applications.

There are several other software frameworks available for the creation of VR applications, ranging from low-level scene graph implementations to complete toolkits. These include Iris Performer [Rohlf & Helman, 1994], OpenSceneGraph [Web-OpenScenegraph] or OpenInventor [Wernecke, 1994].

Our Software Approach

The problem that no suitable VR libraries existed, and the lack of software standards motivated us to create an environment for rapid virtual reality application development on the Responsive Workbench. It resulted in the RWB Library and the RWB Simulator [Koutek & Post, 2002]. This library [Web-RWB-lib] provides the basic functionality for a virtual reality application on the RWB, including stereoscopic projection, tracking support, user interaction, object manipulation, collision detection, etc. Written on top of Performer and OpenGL, the RWB Library still gives the user access to the complete functionality provided by Performer or OpenGL, see Figure 3.14.

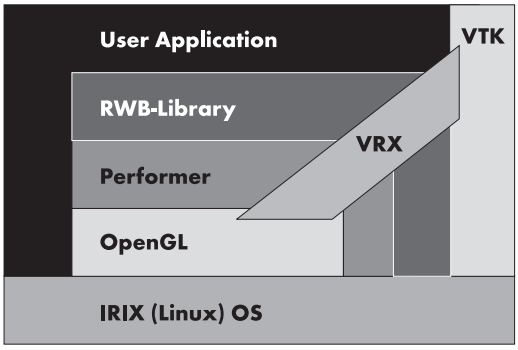


Figure 3.14: Software architecture of RWB applications

Iris/OpenGL Performer [Eckel *et al.*, 1997] is a 3D rendering toolkit for developers of real-time interactive graphics applications. It provides efficient access to the features of the Onyx2 such as multiprocessing on multiple CPU's, multiple visualization pipelines, shared memory, etc. Performer is built on top of OpenGL [Web-OpenGL], which provides the application programmer with access to the low-level rendering capabilities of the graphics cards.

Performer provides a good basis for creating a graphical content of a VE, but building an interactive VR application can still be a time-consuming and difficult task. Therefore the RWB Library significantly simplifies writing VR applications for the RWB and adds the "VR functionality" to the graphical objects.

Although the performance of RWB Library applications is generally very good, the developers of applications should still be aware of the underlying Performer and OpenGL techniques during the design process. The structure of RWB applications has to correspond with the Performer multiprocessing scheme for maximum performance.

The problem of developing and debugging applications has been addressed with the RWB Simulator, see Section 3.5. The RWB Simulator is a very useful tool and provides the developer with a birds-eye view on a simulated Workbench which runs the actual application. The ability to play back a recorded Workbench session in the simulator helps the developer during testing, debugging, previewing and creating still-images and movies of existing RWB applications, see Figure 3.25.

On top of the RWB Library we have developed the Virtual Reality eXplorer (VRX), our data exploration toolkit for VR, which has an interface to *VTK* (Visualization Tool Kit) [Schroeder *et al.*, 1999]. VRX contains a set of exploration, visualization and probing tools for more efficient data analysis in the VE on the Workbench, see Section 5.2.

3.4.2 Performer and Scene Graph Basics

Performer is a scene graph-based 3D rendering toolkit [Rohlf & Helman, 1994]. This concept allows the creation of complex hierarchical graphical scenes, while the system automatically takes care of most complex geometric transformations, memory management and performance optimizations. Detailed technical information can be found in the Performer documentation [Eckel *et al.*, 1997].

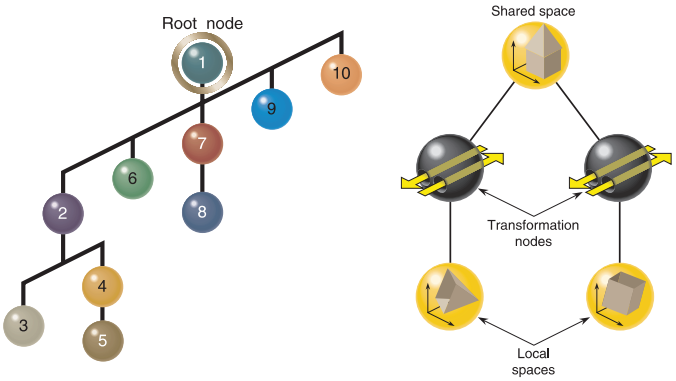


Figure 3.15: Illustration of Performer scene graph: the hierarchical traversal order (left) and coordinate transformations (right) [Image source: Perf. manual]

The content of the virtual world is stored in a hierarchical scene graph, see Figure 3.15. A node of the graph may hold, for example, the data for a geometric primitive or a local coordinate system (**pfDCS** - Performer Dynamic Coordinate System). Different types of nodes provide mechanisms for grouping, animation, level of detail and other functional concepts. The scene graph is traversed at several stages (by scene traversals) which together form the rendering pipeline (Figure 3.16). A typical rendering scheme consists of three processing stages:

- **APP** updates the location and the look of geometries and it updates also the viewing frustum.
- **CULL** determines which geometries in the scene are visible (in the viewing frustum) taking occlusion into account.
- **DRAW** creates OpenGL graphical instructions for rendering the scene.

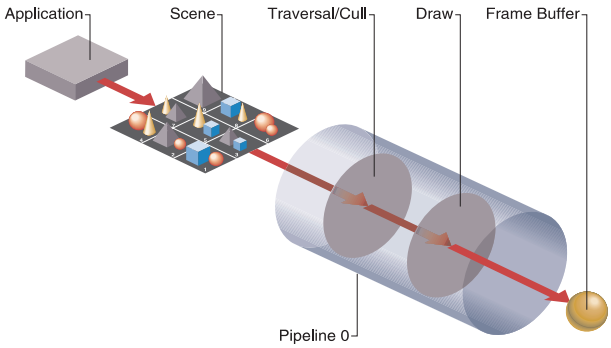


Figure 3.16: Performer rendering pipeline [Image source: Performer manual]

Both the APP and CULL stages traverse the entire scene graph. In the APP stage, changes are made to the nodes in the scene graph. The CULL stage traverses these nodes, calculates the coordinate transformation of all nodes and constructs a display list of visible geometry to be rendered. The DRAW stage transfers this list to a set of OpenGL graphical instructions. Finally, these instructions are sent to the graphics hardware to render the visible geometry. As a result, the changes made to the scene graph in the APP stage are only made visible after a period of time (the latency, discussed in the previous section) which it takes to complete all other stages.

For maximum performance, each of these stages can be executed in parallel as a separate process on a different CPU (Figure 3.17). When using three CPU's, three frames can be processed in parallel, resulting in a considerable increase of performance over serial execution. More time is available for each stage while maintaining the same frame rate.

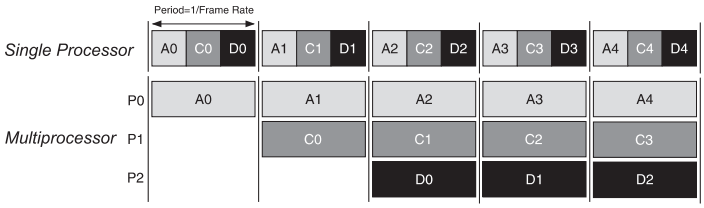


Figure 3.17: Multiprocessing scheme of Performer

The communication between the stages and processors takes place through the Shared Memory Arena, a memory region that can be accessed by the different processes (Figure 3.18). For every frame in the rendering pipeline a unique copy of the scene graph is stored in shared memory. Each of the three stages works on their copy of the scene graph, which guarantees data consistency during processing of the stages.

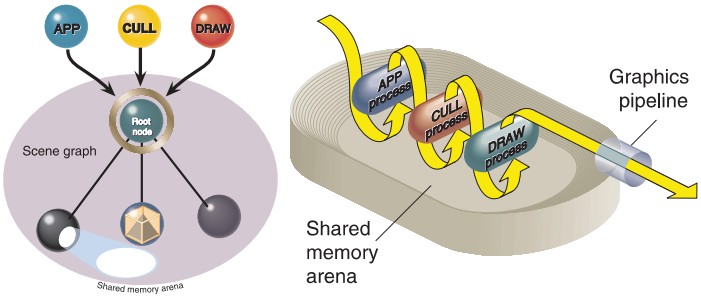


Figure 3.18: Shared memory usage by the rendering stages of Performer

Although the Performer scene graph is copied to maintain data consistency between stages, only one shared copy of the data exists that contains the raw geometry and appearance information (e.g. vertices, colors and surface normals). If this aspect is disregarded, for example the APP stage can update the raw geometries or the colors while the DRAW stage is reading the same data concurrently, leading to unpredictable effects on rendered geometry. We use Performer's solution (pfFlux) to cope with this. The pfFlux node is a container for dynamic data that enables multiprocessed generation and use of geometrical data in the scene graph. A pfFlux node is stored in shared memory and internally consists of multiple buffers of data, allowing multiple processes to have their copy of the data on which they are working. As a result, one process that is busy creating for example a list of vertices in one buffer will not interfere with another process that uses another buffer in the same pfFlux node (e.g. which contains the most recent created display list) to render the geometry.

This multiprocessor-safe buffering technique allows asynchronous geometry data updates from the main Performer processing stages. This is especially useful in cases where intensive calculations slow down the APP stage and as a

result slow down the overall scene update rate. Such intensive calculations can be performed asynchronously in a separate process on a different CPU, using the pfFlux buffer to store the results. As we use just three of the four available processors of the Onyx2 for the APP, CULL and DRAW stages, we have reserved the fourth processor for an asynchronous COMPUTE process to perform computationally intensive operations. The time critical Performer stages are now relieved of the calculation and can simply use the latest available buffer of the pfFlux node. We use this scheme for the update of all visualization tools in our VRX framework (see Section 5.2).

Stereo with Performer

Figure 3.19 shows a scheme of the quad-buffer stereo approach that we have implemented using one graphics card (SGI Infinite Reality 2). Two pfChannels are used to view the scene for the left and the right eye and one pfPipe is rendering the images in an alternating fashion. We use a resolution of 1120×840 pixels at display refresh rate of 96 Hz (48 Hz for each eye). DRAW traversal of the scene graph and consequently rendering is triggered at 24 Hz.

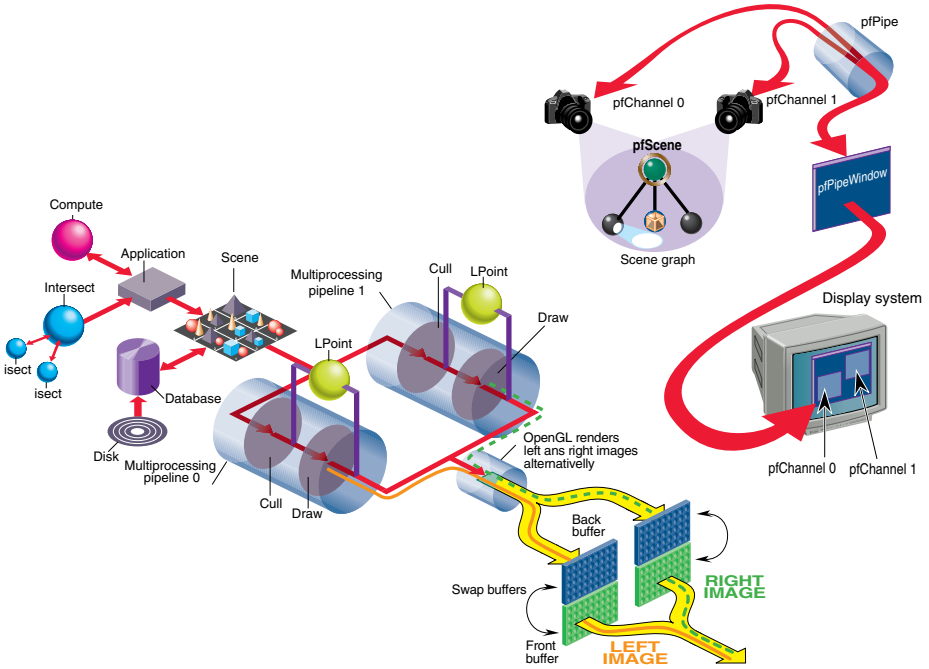


Figure 3.19: Single graphics pipe stereo with left and right pfChannels

As described in Chapter 2, a VR application requires a more specific approach than high performance graphics only. The RWB Library is a VR extension of the real-time 3D graphics, provided by Performer and OpenGL.

3.4.3 Structure of RWB Library Applications

The RWB Library provides an abstract layer on top of Performer, taking care of all required aspects of VE and RWB specifics, such as the creation of the off-axis perspective stereo projection, calibrated tracking of input-devices, and multi-processing. The library provides a ready-to-use VE, optimized for the RWB. This library has an object-oriented structure, allows development of new user-functions, and it forms a basis for programming VR applications.

A simple VE that supports head-tracking, stereo-perspective viewing, and object interaction can be constructed by only a few lines of code (see code fragment below). The VE can be filled by adding graphical objects (generally of *pfNode* type) to the scene graph under the root node of RWB applications, called *App-worldDCS*. These graphical pfNodes can be created by using Performer functions. By manually defining the properties of geometrical primitives (*pfGeoset*) such as color, shape and texture (e.g. pfMaterial, pfTexture) the appearance of graphical objects can be defined. The graphical nodes and whole sub-scene graphs can also be imported from geometry files. The added objects will appear correctly in the virtual environment, but the user cannot interact with them. For this purpose the RWB Library provides the *RWB-Object* class and the *RWB-Interactor* class.

The following simple example of an RWB application begins with an initialization of the RWB Library, building the scene graph, setting up the stereo-view channels and initialization of the RWB Simulator. The user application defines user-code functions: synchronous with main APP process and asynchronous process. The user-exit function must be also specified because after starting the RWB-main-loop the program never returns to the calling function.

```
                                        // *** main.C,  a simple RWB-LIBRARY example
#include "rwblib.h"                     // *** include RWB Library

int main (int argc, char *argv[])
{
 rwbInit_main(argc,argv);               // **** RWB initialization
 rwbInit_scene();
 rwbInit_view();                    // *** void *arena - contains Performer Shared Arena
 rwbInit_sim();

  Shared->UserCodeFunc          =&My_UserCodeFunc;
  Shared->UserCodeAsyncFunc     =&My_UserCodeFunc_ASYNC;
  Shared->UserExitFunc          =&My_ExitFunc;

 My_InitFunc();                         // *** user program initialization
 Show/Hide_GlobalCoordXYZ();
 MakeDefaultPadPanel();                 // .. creates a blue semi-transparent Plexipad
 Show/HideGround();

 rwbForkMain();                         // **** starts the main-loop of the RWB application

// *** The program never returns here, user must use Shared->UserExitFunc !
}

void My_UserCodeFunc() {..}         //** this function is re-called from the main loop
                                    //    in the APP process, it may modify the scene graph
```

```
void My_UserCodeFunc_ASYNC() {..} //** this function runs in an asynchronous process
void My_ExitFunc() {..}          //** user exit function to clean-up things...

void My_InitFunc()               //*** initialization of the user application
{
 pfNode *my_grNode;              // *** pfNode can be any performer sub-scene graph

 my_grNode = CreateGRobj( pfdNewCube( arena, 0.f,0.f,1.f ) ); // *** Create a blue cube
 my_grNode->setName("BlueCube");

 Shared->App_worldDCS->addChild(my_grNode);
 // *** directly attacht to the root of scene graph "Application World DCS"
 // ... or better via rwbObj...

 rwbObj *obj;     obj= new rwbObj(RWB_BV_AABOX);
 // *** RWB_BV_AABOX, RWB_BV_ORBOX .. axis-aligned/oriented bounding box,
 // *** RWB_BV_SPHERE                .. bounding sphere

 obj->EnableCollisions();       //  obj->DisableCollisions();

 obj->attachGRobj(my_grNode,x,y,z,sx,sy,sz,h,p,r);
 // attaching pfNode to rwbObj at position (x,y,z), size (sx,sy,sz), rotation (h,p,r)

 obj->add_to_SceneGraph (Shared->App_worldDCS); // where should this rwbObj be placed

 obj->ShowBoundingVolume();     //   or use obj->HideBoundingVolume();

 obj->visible_selection=1;      // *** selected objects are highlighted with red color
 obj->selectable=1;             // *** RWB-Object can be selected
 obj->manipable=1;              // *** RWB-Object can be manipulated

 return;
}
```

The actual application can be programmed either via the user-code functions, or with traversal callback functions (APP, CULL, DRAW, ISECT) of any *pfNode* that is attached to the scene graph, further using RWB-Objects and their callback functions, or by different RWB-Interactors.

RWB-Object Class

The basis for interactive virtual objects in the VE is provided by the RWB-Object class (*rwbObj*). This class adds several ready-to-use functions to regular Performer graphical nodes, such as support for selection, manipulation, object hierarchy, collision detection and handling of constraints.

Figure 3.20 shows the basic structure of RWB-Objects. RWB-Object has a structure of a sub-scene graph ending with a graphic pfNode, which has to contain some geometry. It can be directly a pfGeode node with geometrical primitives in pfGeoSet(s). But it can be also another sub-scene graph that ends with pfGeode. RWB-Object controls several transformation nodes: *DCS* for position and orientation of the object, *zoomDCS* for uniform scaling (used for zooming onto objects) and *scaleDCS* that is used to scale the attached pfNode, containing the geometrical objects.

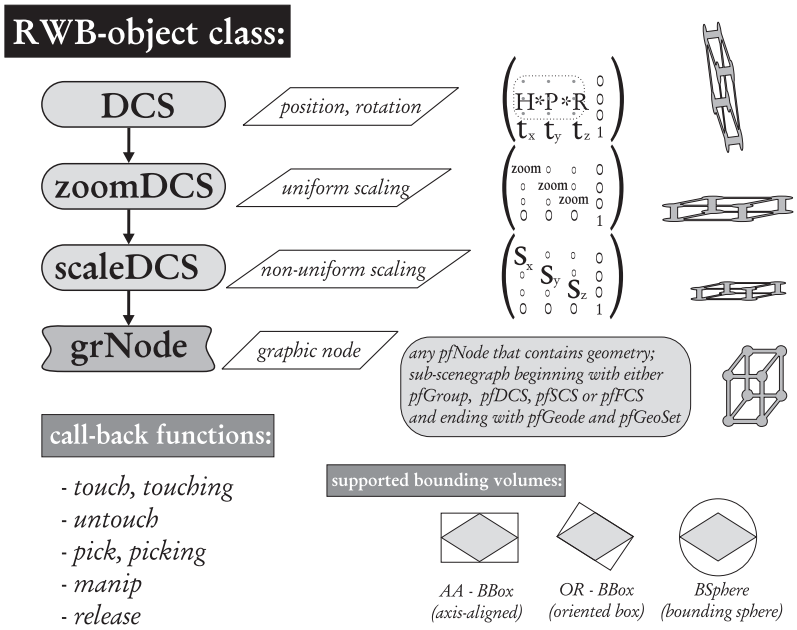


Figure 3.20: RWB-Object class structure

Performer offers SCS, DCS and FCS transformation nodes (Static, Dynamic and Flux Coordinate System). These nodes in fact represent a "packed" transformation matrix: $scaleMatrix \times rotationMatrix \times translationMatrix$. This is a rather effective representation but the dependency of scaling, rotation and translation makes it difficult to implement the flexible scene graph architecture that we need for interactive virtual objects. Therefore we have separated the scaling transformations from rotation and translation. A big advantage of this solution is that we can easily add child RWB-Objects under any DCS node of a parent RWB-Object without any problems with the child object's scaling.

Another problem, which is shown in Figures 3.21 and 3.22, is the problem of uniform and non-uniform scaling. The Performer transformation matrix routines are restricted to affine, orthogonal and orthonormal matrices and facilitate efficient matrix multiplications. An affine transformation leaves the homogeneous coordinate unchanged; that is, in which the last column is (0,0,0,1). An orthogonal transformation is one that preserves angles. It can include translation, rotation, and uniform scaling, but no shearing or non-uniform scaling. An orthonormal transformation is an orthogonal transformation that preserves distances; that is, one that contains no scaling.

As we needed to implement also non-uniform scaling, we had to separate the scaling transformation of the underlying graphic node (*grNode*) with the object geometry (i.e. make a box with dimensions of 4x10x2 when the standard Performer functions for creating primitive objects produce dimensions of 1x1x1).

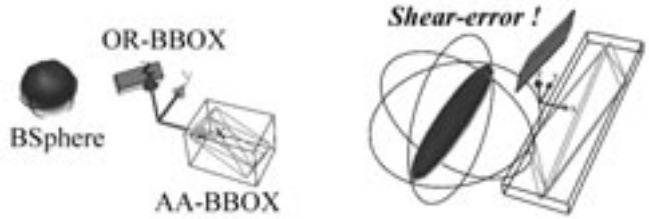


Figure 3.21: Original scene containing graphical RWB-Objects with appropriate bounding volumes (left); When these objects are placed under a non-uniformly scaled and rotated pfDCS node then a shear-error is introduced into the scene graph (right).

The RWB-Object class also implements all basic operations with bounding volumes (BV) like automatic updating of BV during object manipulations and transformations, object selections via BV, and collisions between BVs of other RWB-Objects. Basic BV types are *axis-aligned or oriented bounding boxes,* and *bounding spheres.*

First, RWB-Objects need to have attached a geometry node (*grNode*). The user must specify dimensions, orientation and position in the parent coordinate system. After that, a bounding volume is computed and, in case of enabled collisions, also the triangle collision cache is created. It is also possible to enable and put constraints on object manipulation. Using a constraint mask can be any degree of freedom (xyz, hpr) enabled or disabled. For more information on collision detection and constraints see Section 4.1.3.

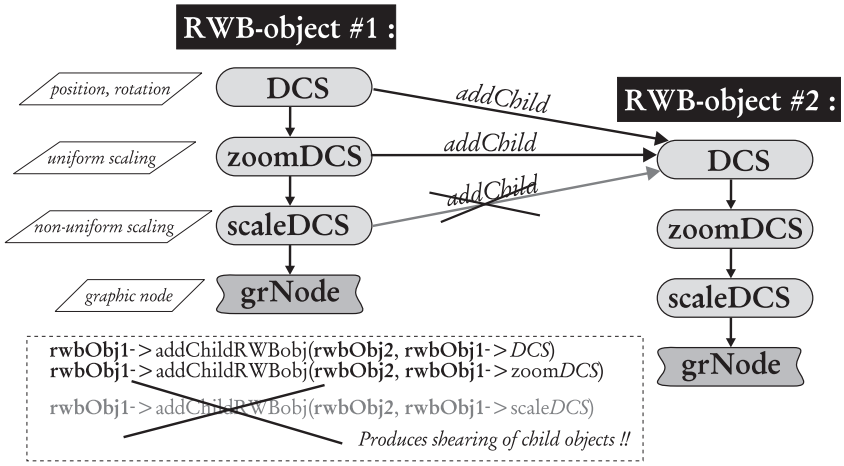


Figure 3.22: Linking RWB-Objects: adding a child to a parent rwbObject

To make the RWB-Object active and visible in the virtual environment, it has to be added to the scene graph, usually under the App-worldDCS node. It is also possible to link RWB-Objects together (Figure 3.22). To prevent the shearing effect (Figure 3.21) in the child object, it should not be added under scaleDCS of

the parent RWB-Object. This "unexpected" shearing appears when rotation is performed in a non-uniformly scaled coordinate system.

Behaviour and interactivity can be implemented into RWB-Objects via callback functions (for example, when the stylus is inside the BV of the object, it can resize the object, or when an object is selected, the pick callback changes its color to red). These functions are triggered from RWB-Interactors.

Moreover, new classes can be derived from the *rwbObj* class, allowing rapid development of virtual objects with custom-made behaviour. We used this technique to construct RWB-Widget classes. These interaction widgets, such as buttons and sliders, are derived from the generic *rwbObj* class. In addition, the RWB-Widget classes again can be used for other derived classes, for example to represent a specific type of sliders, such as the slider with three value controllers used in the colormapper widget (see Section 5.2).

RWB-Interactor Class

An important aspect of a VR application is interaction. Through the use of RWB-Interactor classes various interaction scenarios can be implemented. In these scenarios the behaviour of the application in response to user actions with input devices is described. In Section 5.1 several interaction methods and examples are described in detail.

The RWB-Interactor reads the input devices and determines the state of the user-interaction, see Figure 3.23. If an event is detected the RWB-Interactor triggers the appropriate actions and changes the current user-interaction state. For example, if the stylus has moved inside an *rwbObj*, the RWB-Interactor can trigger a method of this object in order to highlight its selection, while the interaction state is set to "selecting object". The library uses a default RWB-Interactor, which implements both direct and ray-casting selection and manipulation of virtual objects. There are two interaction modes: direct stylus interaction without visible ray and with enabled ray-casting. Switching between these two modes is done by clicking the stylus button when no object is selected.

When only direct interaction is enabled then selection of objects is based on checking whether the stylus is inside the bounding volume of an RWB-Object that is attached to the scene graph. Sometimes it happens that the stylus position is inside more than one RWB-Object at the same time. In such a case, the object closest to the stylus is selected (when it is "selectable"). The BV intersection tests are very fast. We keep a list of all active RWB-Objects in the virtual environment and their BVs.

The ray-casting selection is a more time-consuming operation. We use the ISECT traversal process of Performer to check all ray intersections with all object geometries in the scene. First the objects are checked whether the ray actually hits their bounding volumes (BBOX or BSPHERE) before all their triangles are checked for a ray intersection.

Even when the ray is activated, the direct interaction has a higher priority. Thus, when the stylus moved into some bounding volume (its object must be "selectable") then the ray-casting selection is temporarily disabled. When the user moves the stylus outside the selected object, the ray-casting status is restored. If there is no direct selection then the ray-casting intersection procedure is activated.

We distinguish several events, to which the RWB-Object concerned can react with callback functions, see Figure 3.23. These events and callbacks are:

touch event - stylus is moved inside of the object, or the ray has intersected it;

touching event - object remains intersected and stylus button is not pressed;

untouch event - stylus moves away from the object;

pick event - at the moment of the first click on the stylus button;

manip event - the object is "manipable", an object manipulation is performed;

picking event - stylus button is pressed, object can not be manipulated;

release event - stylus button is released and object manipulation stops;

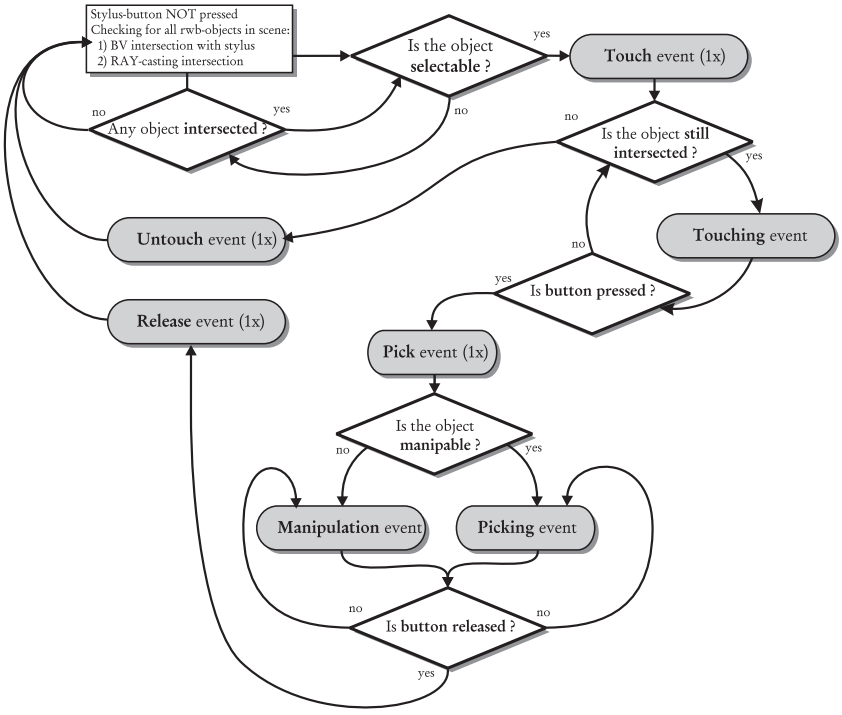


Figure 3.23: Interaction-handling routine of the RWB-Interactor

Besides this standard interaction scenario, custom interaction scenarios can be defined to create new interaction methods. Using the generic RWB-Interactor class we can easily define new interaction tools, such as the zoom tool, the ROI tool or the Spring Manipulator, see Section 5.1. We have implemented two interactor schemes:

The user extension of RWB-Interactor first executes the default RWB-Interactor to perform the intersection checks and to obtain a new interaction state and interaction points; then the user-interactor function defines the desired behaviour *(examples: ZOOM, ROI, Click-ISO, Probing Tools*, see Section 5.1).

The fully user-defined RWB-Interactor overloads the functionality of the default interactor and processes itself the input sensors (via RWB-Lib) and implements the desired functionality *(examples: Spring Manipulators, Virtual Particle Steering*, see Section 4.2).

The user RWB-Interactors can be activated by clicking on a widget button. The RWB-Interactor can be deactivated by the tool itself or by activating another interactor. For example, the zoom tool is automatically deactivated after a zoom action and control is returned to the default RWB-Interactor. The design of the interactor determines the way the tool works. The event-based interactors, straightforward coordinate transformations between the Plexipad and stylus and the simulator environment, all provided by the RWB Library, allow rapid development of new (two-handed) interaction tools.

The following pseudo code fragment illustrates how the interaction states are used in the *handleInteraction* function of a *Zoom-Interactor* (user extension of RWB-interactor).

```
// *** Zoom interactor pseudo code, main interaction loop
void Zoom_Interactor::handleInteraction()
{
switch (status) {                                          //** select state
 case Zoom_RESET:
        reset();
 case Zoom_NOTHING_SELECTED:                               //** state nothing sel.
   if (!globalInteractor->stylus_button)                   //* check button
   { /* do nothing */}                                     // button not pressed
   else                                                    // button pressed
   { status=Zoom_WAIT_FOR_BUT_RELEASE;                     // change state
     startZoom();                                          // start the zoom
   }
 case Zoom_WAIT_FOR_BUT_RELEASE:                           //** state zooming
   if (globalInteractor->stylus_button)                    //* check button
    {                                                      // still pressed
    updateZoom(Shared->cur_interactor->interactionPoint); // update zoom
    }
   else                                                    //* button released!
    { status=Zoom_NOTHING_SELECTED;                        // change state
      stopZoom();                                          // stop the zoom
   }
}
```

The behaviour of an application is fully defined by the combination of user-defined RWB-objects, RWB-Interactors and the general user-code. This can contain anything, from general application control to a specific animation of a virtual object, and is executed every frame in the APP traversal. A more elegant way of defining specific application behaviour is to supply user defined callback functions to the various nodes. These callback functions can be automatically executed when the node is visited during a scene graph traversal in one of the processing stages. For example, an APP callback function, which rotates an object, can be executed automatically every time the object is visited during the APP traversal.

The Main Loop of an RWB-Application

A brief overview of the way RWB-Applications work is shown in the following code fragment. After the initialization of the VE, the RWB Library initiates Performer's scene graph traversal and starts the main loop in the APP process.

```
// ***** main_loop of RWB application (APP-process):

 while (!Shared->exitFlag)
 {
    if (RWB_SIMULATOR) handleSIM_navigation(); // *** in case of RWB-Simulator
                                               //     handle the simulator-navigation

    //*** wait until the next frame boundary, synchronization of all performer processes
    pfSync();

    // *** NOW we do TIME CRITICAL OPERATIONS

    UpdateUserInput();     //** reads new tracker_data, updates stylus DCS
    UpdateHeadCoords();  //** update head_positions + view-position

    if (Shared->UserCodeCritical!=NULL) Shared->UserCodeCritical();
    //*** calling user time-critical code, if it exists...

    if (Shared->drawStats) Shared->rwbview->left->drawStats(); //** draw channel statistics

    //*** initiate traversal using current state
    //*** synchronized start of APP, CULL, DRAW, DBASE, ISECT and COMPUTE traversals/process
    pfFrame();

    Shared->cur_interactor->handleInteraction();
    //*** handle interaction with application via RWB_Interactor

    if (!Shared->rwbOptions->ForkedXInput) HandleEvents(); //** handle X events

    HandleEventsInMainLOOP(); //** handle events that can modify the scene graph

    if (Shared->UserCodeFunc!=NULL)
      if (!Shared->stop) Shared->UserCodeFunc();  //** call the synchroneous user code

    UpdateSceneGraph(); //** make changes to the scene graph

 } //** end of main-loop
```

At the beginning of the main loop the APP process waits for all other processes to finish the last frame (*pfSync*). After this the library performs time critical functions such as reading the tracker information. This information consists of the position and orientation of all tracking sensors. In the real RWB application this data is read directly from shared memory of the tracker daemon while in the RWB Simulator this data is read from a stored tracking-data file. Then, all other processes (e.g. CULL and DRAW) are started synchronously (*pfFrame*). After this, the currently active RWB-Interactor is invoked. This interactor evaluates the status of the input devices and triggers the appropriate actions (e.g. move a virtual object to a new stylus position). Next, the user-defined or application-specific code is executed.

Finally, the main loop waits until all stages have finished. The updated scene graph from APP is then passed to the CULL stage. This stage eliminates the invisible elements and sorts the graphical objects in a display list of geometry. This list is passed to the DRAW stage. In this last stage the resulting display list is passed to the graphics pipeline which renders it to the screen. If all stages are finished and data has been transferred to the next stages, the loop starts again from *pfSync.*

The combination of the RWB Library and Performer allows a rapid development of VR applications and interaction techniques for different application areas. The use of multiprocessing techniques provides performance for high interactivity even with complex VEs.

3.4.4 RWB Library Performance

The performance of RWB applications can be characterized by the update and rendering rate of the scene and the program latency, which is the time spent between the user actions and the resulting system feedback.

We use three parallel rendering stages (APP, CULL, DRAW), and the update rate is determined by the slowest stage in the rendering pipeline. The latency is therefore three times the time spent in the slowest stage. A frame rate of 5Hz is caused by the 200 ms spent in one of the rendering stages. The latency caused by the rendering pipeline is here approximately 3 * 200 ms = 600 ms. The higher the scene update frequency, the lower the latency is and the more realism and comfort is experienced by the users. Although an update frequency of 10Hz is experienced as sufficient (latency of 300ms), rates of 20 Hz and higher are found most comfortable (latency of 150ms). In our setup the frequency is limited to the maximum refresh rate of the CRT projector (for the chosen resolution of 1120*840) and the shutter glasses (96 Hz, left/right image = 48 Hz). The main loop of our visualization system runs at 24 Hz. With too complex calculations and large geometric models the frame rate can drop under 10 Hz. Therefore we try not to generate very complex models (less than 100.000 triangles in the whole scene).

The scene update frequency is reduced by the time consumed by operations that have to be completed each update. These operations include the reading of the tracker data, execution of user-defined application functions (e.g. filtering data), constructing of the graphical primitives in the visual scene and finally rendering the primitives to the graphics frame buffer. The more complex the user-defined calculations are, and the more complex the generated graphical scene is, the lower the scene update frequency will be. The key towards a high update frequency is to exploit the computational power from the available computer system(s) and to reduce complex calculations, minimize number of graphical primitives, and use simplified and LOD (level of detail) geometric models.

3.4.5 3D Interaction and User Interface

The RWB Library offers user interaction with devices such as a keyboard, a mouse, a spacemouse (a kind of 6 DOF joystick), a stylus (6 DOF) and a Plexipad. The spacemouse can be used for navigation in large environments and to its 9 keys extra functionality can be assigned by the application. For spatial (”real 3D”) interaction the RWB applications use the Plexipad and the stylus with a tracking sensor and one button.

RWB Library also provides a basic 3D widget set for building a 3D user interface with buttons, sliders, menus, or displays, dial and type-in windows. At the position of the stylus the actual tool can be displayed. The full spectrum of 3D user interface facilities is demonstrated in the case studies in Chapter 6.

Monitoring of User Interaction

A very important aspect of working with the Responsive Workbench is to be able to monitor and debug execution of an RWB application in a distributed application-development environment.

We were mainly concerned with our VR and visualization laboratories that were until the year 2000 equipped almost exclusively with SGI graphics workstations. Our intention was to use other workstations (one Onyx, one Octane and several O2s) in combination with the VR facility (i.e. the RWB), which is driven by the Onyx2. Later, we have also included PC graphics workstations into our application development environment.

We have incorporated in the RWB Library a monitoring function for the user interaction. The principle is quite simple. During the execution of the application on the Workbench the tracker data and application time-stamps are written into a file (or sent through a network) for immediate or later use, such as animated replay of a Workbench session in the RWB Simulator, see Section 3.5.

3.5 RWB Simulator: A Tool for Application Development and Analysis

The RWB Simulator is a desktop development environment for RWB Library applications. This tool provides application support in areas of: *development, evaluation, learning, presentation and navigation assistance.*

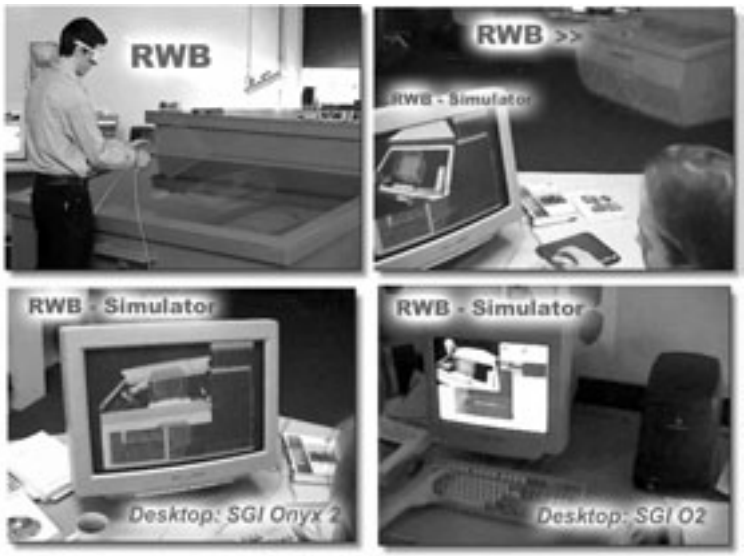


Figure 3.24: Our VR laboratory with the Responsive Workbench and RWB Simulator platforms

3.5.1 Motivation

The process of developing a VR application needs special attention. With some systems the development has to be done directly on the VR facility. Some systems have an option of creating the VR applications off-line in a sort of "VR on desktop - simulator", like the CAVE Simulator [Web-CAVElib]. There are also systems such as Avango [Dai *et al.*, 1997], Lightning [Bues *et al.*, 2001] or VR-Juggler [Bierbaum *et al.*, 2001], where the application developer can define the virtual application platform to be, for example, a desktop system driven by keyboard and mouse which emulate the 3D input from the immersive VR systems.

However, trying to simulate a 6 DOF (degrees of freedom) input device with a 2 DOF mouse is not very practical. It is impossible to simulate in this way a complex 3D interaction that the user does (would do) in the virtual environment with the implemented application. We think that user interaction is one of the most important aspects of VR applications. Therefore we should devote enough attention to it during the development of an application. The best way

of testing the spatial interaction is in the immersive VE itself. Development is done much easier off-line on a desktop-system. The developer then pays the price of missing 3D interaction with the application, which can be compensated by using (quasi-) 3D input devices like the spacemouse, but still that is not the same level of interaction as provided by the Responsive Workbench or in other immersive VEs.

Our proposed solution for this problem is based on capturing the user's (developer's) 3D interaction with the prototype application on the Responsive Workbench. We measure the tracker data (tracking of user's head and 3D interaction devices) with a given frame rate and we also store time-stamps of the RWB application running on the Responsive Workbench. This measured data is then used by the application developer within the RWB Simulator for the next stage of development and improvement of the RWB application. The RWB Simulator works on a desktop system and is controlled by mouse and keyboard commands. The tracker data represent the interactions of the virtual user with the application. A powerful tool is available to the developer for tracing and simulating of the application forward, in some cases also backward in time, with captured 3D interaction.

We try to solve the problem of simulating the spatial interaction during VR application development with our concept of the RWB Simulator. Also, the use of this tool has a strong impact on the processes of application development, analysis, evaluation and presentation. It reduces the time needed on the Responsive Workbench and increases the efficiency of the whole process.

3.5.2 Development of RWB Applications

Development and implementation of VR application should be efficient. We try to minimize the time spent in immersive VE during development, because it is not always convenient and effective to debug or monitor an RWB application on the RWB itself. Sometimes the user performs application-specific tasks, and it is difficult to see if the task or the underlying algorithm works properly, when the user is just standing at the Workbench and wearing the shutter glasses. Usually, program variables will be written onto the screen and analyzed. On the real Workbench we cannot always easily pause, slow down and debug the application. This becomes even more complex if we consider the multiprocessing nature of the RWB application.

The RWB Simulator uses the captured tracker and application data and highly interactive applications benefit from it. Time dependent simulations can make use of the unified RWB-time and the RWB Simulator can play back a simulation at any speed, usually at lower speeds, so that the developer is able to check the proper behaviour of the simulated process. Some simulations can even be played backwards in the RWB Simulator. Using our RWB Library and Simulator, the process of application development runs as follows, see Figure 3.25.

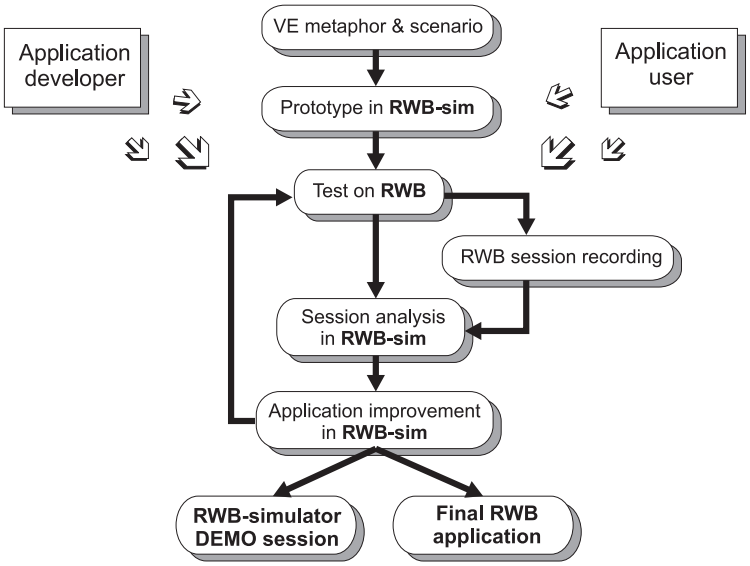


Figure 3.25: Application development scheme

First, there must be a clear idea about the RWB application and its purpose - the application scenario. The proper VE metaphors have to be considered.

The developer prepares a prototype of the application: the scene graph of the virtual world, basic functions and callbacks, the user interface, etc. During this preparation stage the application is compiled in the simulator mode.

In the next step, the user/developer runs the application on the real RWB, and performs some tests and adjustments. The developer can record tracker data for having some user's interaction data, and switches back to the simulator mode. This process repeats until the implementation is finished. At later stages of this iterative process the real users of the RWB application can test the application and the developer can record their Workbench sessions. With this real user data the developer can adjust the implementation and check whether it will fulfill the real users' needs and their interaction abilities without them, because the developer already has their interaction data.

This way we can much better address the problem that usually the best user of an application is the developer. With the real user data the RWB application can be really "tuned" for the user and not the developer. We still have to learn a lot about this user-developer cooperation process, but we think that this might be a good direction.

After the final test of the application on the real Workbench the RWB Simulator can be used to produce images and animations for presentations. Also demo sessions can be produced for the application users to learn how to use it. These demo sessions can be previewed with the RWB Simulator version of the RWB application.

3.5.3 The RWB Simulator Usage

The RWB Simulator is in fact a simulation mode of an application that uses the RWB Library. In practice it is a C++ compiler and linker option. There is also a specially compiled RWB-SIM Library with enabled debugging options, and has a lot of extra functions for tracing the RWB applications. The developer thus has two compilations of the application, one optimized for the workbench and the other for the simulator. For the application developer or the user it is very simple to use.

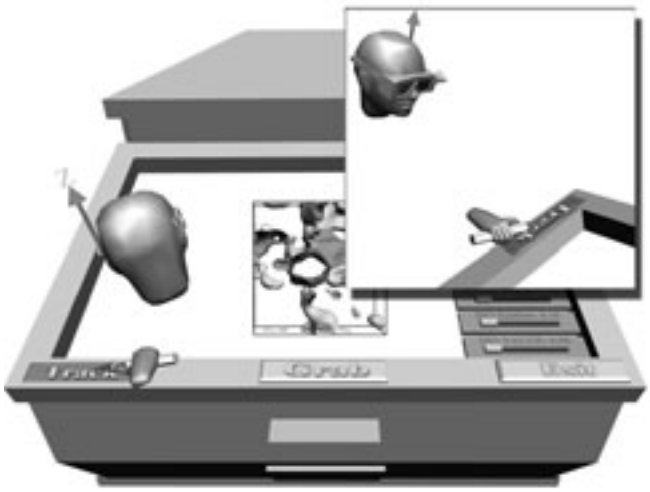


Figure 3.26: A user clicks on the track button to start recording of his interactions with the RWB application. These buttons (track, grab, exit) are invisible on the Responsive Workbench (placed on the wooden bar) and the user gets an audio feedback of the buttons instead of visual feedback when touching these buttons.

In simulation mode, the tracker data are not read from the tracker daemon but from the tracker data file. The application then runs in the same way as on the Responsive Workbench. Stereo is optional in the RWB Simulator on the Onyx2 with the IR2 graphics card. The RWB Simulator can export RED/GREEN stereo images and animations also. On other desktop systems we usually do not provide stereo.

On the Responsive Workbench the user performs the interaction with the RWB and with the running application. The user or the developer (Figure 3.26) can start to capture his interactions with the virtual environment into a file.

Then within the RWB Simulator, the developer sitting at a common workstation can play back what was happening on the real RWB. The application world is displayed on top of a model of the Workbench; the displaying of the

model can be also switched off. The developer can observe the run of the RWB application, how the user performed with it, and the simulator user can navigate around the RWB model with the mouse via a trackball metaphor, see Figure 3.27. The keyboard can be used to control the simulator (e.g. slow, pause, trace back/forward or reset simulation, reposition the user's head or the stylus). In the case when there are no interaction data for the simulator, the 3D interaction can be emulated with keyboard and mouse controls, similar to the CAVE Simulator [Web-CAVElib].

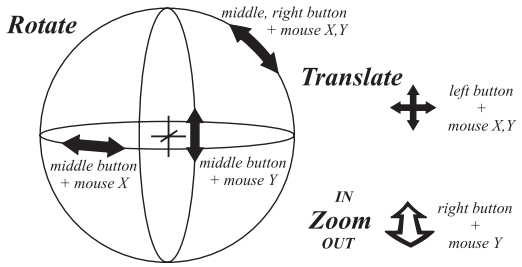
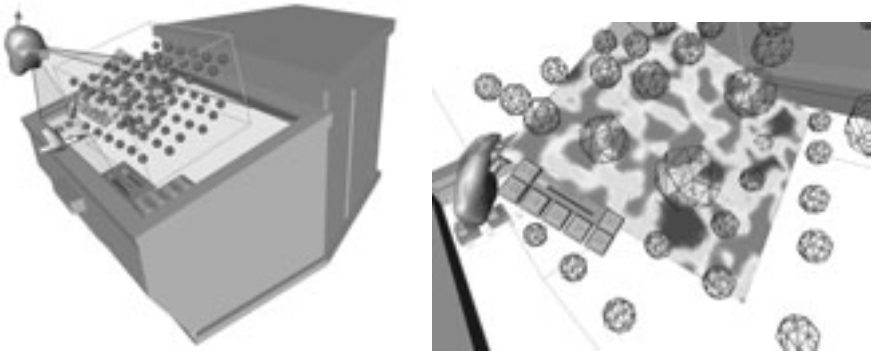


Figure 3.27: RWB Simulator viewpoint navigation via trackball metaphor

In the RWB Simulator the default viewpoint is above the Workbench table overseeing the whole virtual setup with the user and the Workbench, see Figure 3.28(a). It is also possible to show the view from the user's eye position, see Figure 3.28(b).



(a) (b)

Figure 3.28: RWB Simulator: (a) overview with FOV volume; (b) user view

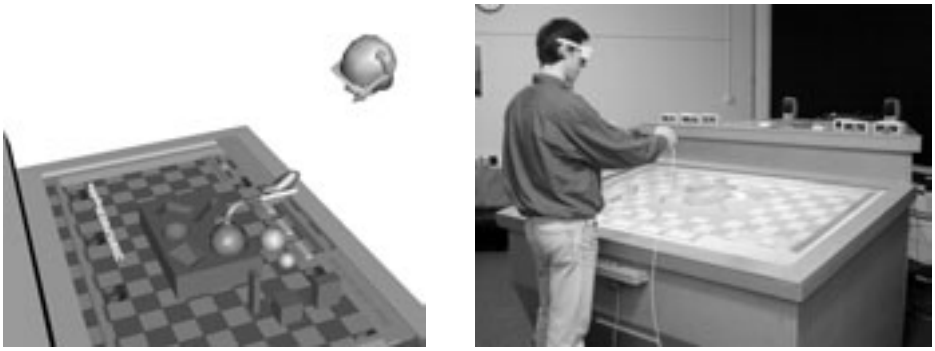
A big advantage of the RWB Simulator is in its portability. The RWB applications can be implemented and developed on common graphic workstations with Iris Performer and the RWB Library. As our VR laboratory was originally equipped only SGI graphics workstations we designed our software for the SGI

systems. The RWB Simulator fully supports SGI Onyx, Octane and O2 family. Logically, the run-time RWB application needs the SGI Onyx 2 with the Infinite-Reality 2 graphics card.

Recently, since Performer is available for Linux, we have also made a compilation of the RWB Simulator for PCs with Linux. The only requirements are properly working OpenGL and Performer, and because the Performer geometry file format (”*.pfa, *.pfb” - Performer ASCII/binary) is not generally used by other applications, it is also practical to install OpenInventor for being able to read the Inventor geometry format (”*.iv”), which is well-known and established in the graphics community.

3.5.4 Presentation of the RWB Application

Another aspect of VR research is demonstration and presentation of results. It is not possible to take stereo/immersive pictures of a user working with an RWB application. Usually, we switch the projection to a monoscopic mode and then we adjust the perspective to align the user with the virtual world, we take a camera and make the photo. The RWB Simulator is very convenient for making pictures/animations of the RWB application (many of the figures from this dissertation are made with the RWB Simulator).



(a) RWB-Simulator: Application overview (b) RWB-photo: Application overview

Figure 3.29: Comparison of pictures: the spring-fork used for manipulation of objects in the mini-world, see also Section 4.2

We have used this system for several research problems and case studies. Most of them will be described in the following chapters. Interactively the most complex RWB application that we have tested with the RWB Simulator was a physics-based world for an assembly task which we have used for designing the spring-based manipulation tools. Object collisions and the dynamic tools were simulated in real time. We have used the RWB Simulator not only in the process of development and design of the tools and their application, but we have also produced a large number of pictures and animations with it.

The reader can compare the images of the same application in the RWB Simulator, see Figure 3.29(a), and the real RWB application photo taken by a camera, see Figure 3.29(b).

Further, RWB Simulator can be used for creating "augmented reality" (AR) still images. A RWB Simulator image is superimposed on a 'real' picture taken by a digital camera. In such case, the photo of the RWB application with the users has to be taken simultaneously with recording the tracker and interaction data. The application can be played back in the RWB Simulator with the RWB photo on the background of the screen, see Figure 3.30. After adjusting the view in the simulator, so that the Workbench model is aligned with the physical Workbench on the photo, we have to replay the interaction session from the RWB and find the moment in time when the photo has been taken. These "AR" still images clearly demonstrate the VR application in the real use.

Figure 3.30: RWB Simulator supports creation of AR still images.

3.6 RWB Library and Simulator Summary

In this chapter we have presented the RWB Library and the RWB Simulator, our environment for the development of RWB applications.

With this system, the applications can be developed using desktop systems as well as the RWB itself. RWB applications can by played back and traced in the simulator environment. The use of the RWB Simulator improves the efficiency of application development. Most of the development time of the RWB application can now be spent on a desktop system.

The RWB Simulator can produce images and animations for presentations. The simulator environment could be also used as an interactive tutor to teach the users to operate the RWB applications.

Chapter 4

3D Interaction in Virtual Environments

In this chapter, we will first give a short overview of existing basic interaction techniques for virtual environments. Basic problems of 3D interaction will be discussed. Interaction methods suitable for the Virtual Workbench, object collisions checking and handling schemes together with constrained manipulation of objects will be presented in Section 4.1.

After this introductory section, we will describe visual force-feedback tools that are based on the spring metaphor, see Section 4.2. The spring-based tools are attached to objects assisting the manipulation. They provide a visual force interface and substitute a real force input.

The spring tools were further adapted for manipulation of atoms in Molecular Dynamics simulations. More about particle steering tools in the MolDRIVE system can be found in Section 4.3.

4.1 Basic Interaction Techniques

Interaction is an essential characteristic of virtual environments. Much has been published about interaction techniques in VR but the quest for truly intuitive and natural interaction techniques is still going on. Interaction between users and virtual environments is complex. Users must be able to navigate through 3D space, manipulate virtual objects with 6 degrees of freedom (DOF), or control parameters of a simulation, and interact with the 3D GUI inside the virtual environment in a user-friendly way.

However, fully 3D interaction is not well understood [Herndon *et al.*, 1994]. Users often have difficulty controlling multiple DOFs simultaneously, performing tasks in a volume rather than on a surface, and understanding 3D spatial relationships. These problems are magnified in immersive VEs. Users are faced with new interaction devices, such as pinch and data glove, tracked wand or stylus, haptic or other devices. The users are expected to quickly learn operating and using them.

Proprioception is a sense of self-awareness of user's body parts, their position and orientation. Proprioception in user interaction within VE has been studied by [Slater *et al.*, 1995; Mine *et al.*, 1997; Mine, 1998]. Similar to the real world where we can use our hands and legs without watching them. This factor is very important for the level of presence experienced in VE.

4.1.1 Interaction Techniques - Overview

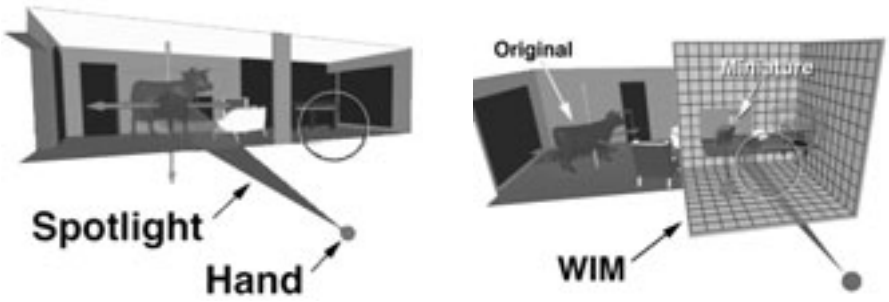
Design of interaction techniques and user interfaces for VR must be done with extreme care. The type of interaction technique usually depends on the task to be performed. In our case, we studied interaction techniques that would assist in visualization and steering of simulations in fully immersive (CAVE, HMD) and semi-immersive virtual environments (RWB). Surveys of interaction techniques for VR can be found in [van de Pol *et al.*, 1999; Bowman & Hodges, 1997a; Bowman & Hodges, 1997b; Mine, 1995].

We can basically divide interaction techniques into three categories: *object selection, object manipulation,* and *navigation in virtual environment.*

We begin with a brief overview of selection, manipulation and navigation techniques in virtual environments. After that we will present our approach of suitable interaction techniques for the Virtual Workbench.

Object Selection

First we will discuss selection techniques. *Direct picking* is the most intuitive and easy way of object selection when the user can reach an object with the pointer. When the object is not within the reach of the user's hand or pointer (stylus), another technique must be used. In *ray casting* (alias *virtual laser pointer*), a light ray (or laser beam) is cast from user's fingers or a pointer, and intersections with objects are evaluated. With simple ray casting the user may find it difficult to select very small or distant objects. A variant of ray casting developed to handle this problem is the *spotlight tool,* see Figure 4.1(a). In *gaze-directed* selection the user can select an object by looking at it. The *pointing* technique allows the user to select an object by pointing at it with a finger or a pointer and an invisible ray shoots out from between user's eyes and pointer. *Virtual hands and go-go techniques* [Poupyrev *et al.*, 1996] give user the possibility to reach distant objects by extending virtual hands further than the user's real hands.



(a) (b)

Figure 4.1: Spotlight selection technique (a); World-in-miniature (WIM) navigation tool (b) [Image source: UNC-CHIMP]

Object Manipulation

Once an object has been selected, the user can manipulate it [Poupyrev *et al.*, 1997; van de Pol *et al.*, 1999; Mine, 1996]. *Manipulation* is a task of changing parameters of a particular object. The number of ways in which an object can be manipulated is almost unlimited. The most important types of object manipulation are: positioning, orientation, scaling, changing an objects' color and shape, creating, grouping and deleting of objects [Bowman & Hodges, 1995].

VE interfaces typically support *direct manipulation,* which allows the user to grasp an object, manipulate it, and then release it. Some systems, usually using VR data or pinch gloves, also support recognition of hand gestures. With *symbolic manipulation,* users manipulate GUI widgets which in turn affect change on objects (i.e. RGB-slider widgets can control a color of an object).

We will now concentrate on object manipulation that controls position and orientation of objects [van de Pol *et al.*, 1999]. When an object is out of reach, *close manipulation* brings this object near to the user. *Popping* brings a distant object into user's hands and after manipulations the object goes back to its position. *Copying* brings a copy of the object into the user's hands, while the original distant object follows the manipulations performed with the copy. *Distant manipulation* allows the user to manipulate distant objects with tools at a distance. In *tele-manipulation,* the user manipulates distant objects just as if they were close to his body. The World-in-miniature (WIM) technique provides users with a handheld copy of the virtual world. The user can indirectly manipulate the objects via their representations in the WIM, see Figure 4.1(b).

Objects can be manipulated in principle by one or two hands by using one or two interaction devices, respectively. Principles of two-handed direct manipulation on the RWB can be found in [Cutler *et al.*, 1997].

Methods for remote object translation techniques for immersive VE, especially the CAVE, are presented in [Mulder, 1998]. Some of these methods can be used on the Responsive Workbench as well.

With the *slave method,* the manipulated object follows translations of the pointer. A disadvantage is that for larger translations the user must perform a series of translate-release actions. The *stick method* connects user's hand or pointer with object by a ray/stick. The object is attached to this stick and follows translations and rotations (center of rotation is the user's hand). With the *3D cross-hair* method, the user can translate an object by dragging the ray of the pointer along one of the cross-hair axes.

For controlling the velocity of objects *fly* or *throttle* methods can be used. With the *fly method,* the direction and the velocity of a pointer is applied to the object. The *throttle method* uses a metaphor of a motorcycle throttle grip. By rotating the pointer about the direction of motion (the direction where points the pointer) the velocity of motion is indicated. Forward and backward motions are derived from the direction of rotation.

[Yoshida *et al.*, 2002] have presented a *scope-based* interaction technique. With the scope-tool and by using two-handed gestures they can select distant objects and manipulate them inside the scope-tool as if they were close.

Navigation in Virtual Environments

Generally, navigation can be regarded as a process of determining a path to be traveled by any object through any environment [Darken & Sibert, 1993]. In the context of virtual environments, this usually means changing the position and orientation of the user's viewpoint. The navigation process can be classified into three distinct categories. *Exploration* is navigation without any explicit target when the user simply explores the environment [van Dam *et al.*, 2000]. *Search tasks* involve moving through the environment to a particular location. Finally, *maneuvering tasks* are characterized by precise movements usually done to position users better for performing other tasks.

The navigation task itself is broken up into a motor component called *travel*, the movement of the viewpoint from place to place, and a cognitive component called *way finding*, the process of developing spatial knowledge and awareness of the surrounding space.

Complete specification of user's movement through the VE needs a direction (goal) of motion and a speed of motion [Mine, 1995]. There are several ways of specifying the direction of motion: *physical walking, hand directed, gaze directed, object driven, goal driven, dynamic world scaling, World-in-miniature (WIM - see Figure 4.1(b)).* The speed control can be basically categorized into: *constant speed, constant acceleration, hand controlled.*

A hands-free technique, *head-directed navigation* for a virtual environment has been presented in [Fuhrman *et al.*, 1998]. It uses head orientation (up and down) to define to backward or forward motion, respectively. Direction of flying through the environment is given by looking left or right.

Redirected walking in place navigation technique for the CAVE has been presented in [Razzaque *et al.*, 2002]. This technique solves a problem of the missing back wall in 4-sided CAVEs, which can seriously disturb the immersion effect, by rotating the world against the rotations of the user's head when the user is walking in place around the virtual environment.

The Personal Interaction Panel (PIP) has been used for navigation [Stoev *et al.*, 2001]. They have developed a *through-the-lens* concept, which provides a scalable preview of the world on the PIP. It makes use of well-known manipulation metaphors: eyeball-in-hand, scene-in-hand, and world-in-miniature.

Navigation techniques suitable for landscape and battlefield visualizations on the Responsive Workbench are briefly described in [Durbin *et al.*, 1998]. *Map-centric navigation* is based on how users interact with a real physical map placed on a table surface. It consists of three navigation modes: *pan, zoom, and pitch/yaw.* The magnitude of the user's gesture controls the distance of the virtual motion.

Another navigation metaphor investigated by Durbin et *al*. is termed *user-centric navigation*. It is loosely based on the metaphor of a user flying above the map as if in an airplane. In this case, the magnitude of user's gesture (hand translation) controls the velocity of the virtual motion.

4.1.2 Interaction Techniques for the Workbench

Considering the specific nature of the Responsive Workbench (laboratory table metaphor, head-tracked central user, available spatial interaction devices, etc.) and the type of VR applications considered, we have chosen to implement the following basic interaction set into the RWB Library.

Object Selection in a VE

The virtual objects on the RWB are usually within reach of the stylus in the user's hand. For selection of distant objects, or objects below the glass plane, we use the *ray-casting (or virtual laser pointer) technique*, see Figure 4.2. For near objects we use *direct picking*, see Figure 4.3.

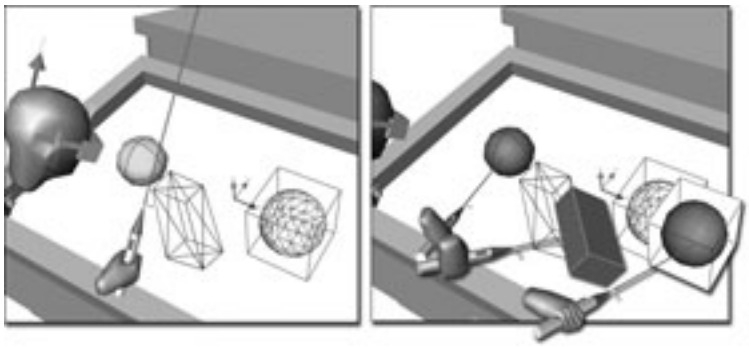


Figure 4.2: Selection of objects using ray-casting technique. The selected object is highlighted by a red color.

Object Manipulation in a VE

In simple cases when an object is within reach of the user's hand the *slave method* is used for object manipulation. Translation and rotation are applied directly to the object, see Figure 4.3.

When the object is distant, the *stick method* combined with the *ray-casting method* is used. Together with ray-casting, the *cross-plane method* is used, where the object is being translated by the ray intersections in a horizontal plane. A similar technique is the *cross-surface method*, where the ray intersects surfaces of other objects (eg. a landscape). These are examples of constrained manipulation.

The ray casting technique has a lower priority than direct interaction with the stylus. The ray is switched off when the direct selection with stylus occurs (Figure 4.3). The user clicks the button and manipulation begins. During object

manipulation our system also evaluates object collisions, as demonstrated on the last position in the manipulation series on this figure.

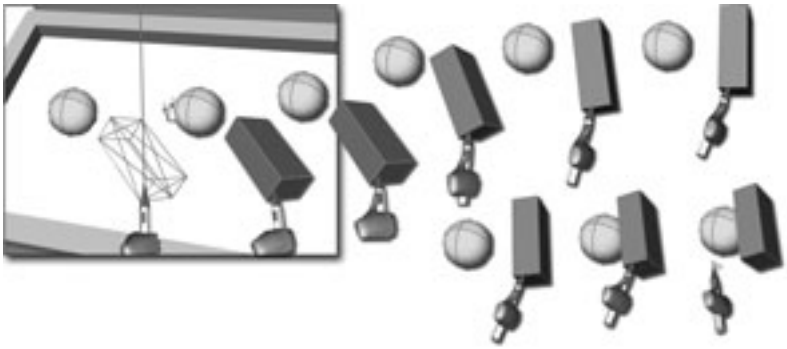


Figure 4.3: Direct manipulation of an object with the stylus.

Navigation in VE

In the RWB Library we have included support for head-tracking so that the user can navigate and walk around the table in a natural way. Further, the user can move (pan), rotate and scale (zoom) the virtual world that is displayed on the RWB.

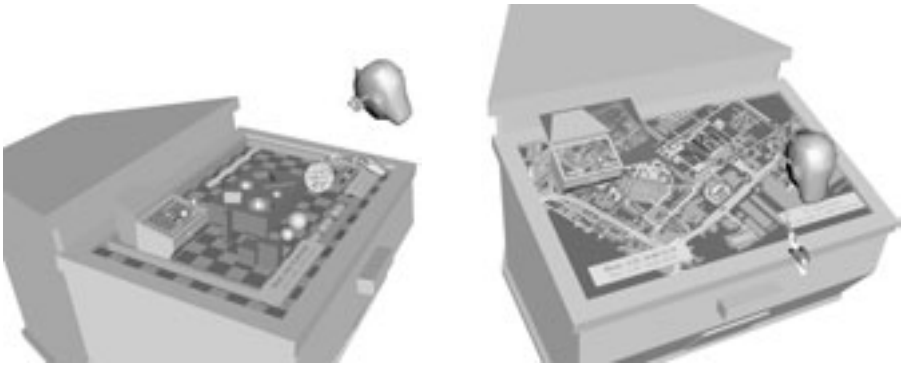


Figure 4.4: Workbench In Workbench (WIW) technique is assisting navigation

To improve the user's orientation in the virtual world we have built in the library the *Workbench In Workbench* technique, see Figure 4.4.

A small copy of the Workbench is displayed on the RWB table top. It contains the whole virtual world as well as the user's head and the stylus. The user can navigate in a large world by looking onto the Workbench miniature and seeing which part of the world is displayed on the RWB. The WIW function helps to locate and manipulate objects which are not projected onto the RWB table top because they are outside the field of view (FOV). These objects are visible in

the Workbench miniature. In the RWB Simulator it can be seen which virtual objects intersect with the FOV, see Figure 3.9. When objects are clipped by the FOV volume the sense of immersion is destroyed. Thus, we have to try to avoid this by placing the virtual objects inside the FOV.

The WIW metaphor can also be used for collaboration with another user in distributed applications. On the miniature of the Workbench the user can see what the other user does in the virtual environment.

4.1.3 Objects Collisions and Object Constraints

Object Collisions

Collision detection is a fundamental problem in computer animation, physically-based modeling, computer simulated environments and robotics [Moore & Wilhelms, 1988]. In these applications an object's motion is constrained by collisions with other objects and by other dynamic constraints [Barzel & Barr, 1988]. The problem has been well studied and described in the literature. However the research on fast and accurate collision detection and handling suitable for VR is still going on.

A survey of collision detection algorithms can be found in [Lin & Gottschalk, 1998]. Collision detection schemes suitable for assembly simulations in Virtual Reality were presented in [Zachmann, 2000]. Several collision detection approaches are available: I-COLLIDE [Cohen *et al.*, 1995], RAPID [Gottschalk *et al.*, 1996], SOLID [van den Bergen, 1999], V-Clip [Mirtich, 1998], and several others.

Most of them use a hierarchical bounding volume (BV) representation (axis aligned and oriented BBoxes, or BSpheres) and space partitioning to speed up the collision checks. Many collision algorithms use BVs to rule out collision checks between objects which are far apart. Usually when an overlap of BVs is detected, an exact "polygonal soup" collision detection algorithm is triggered.

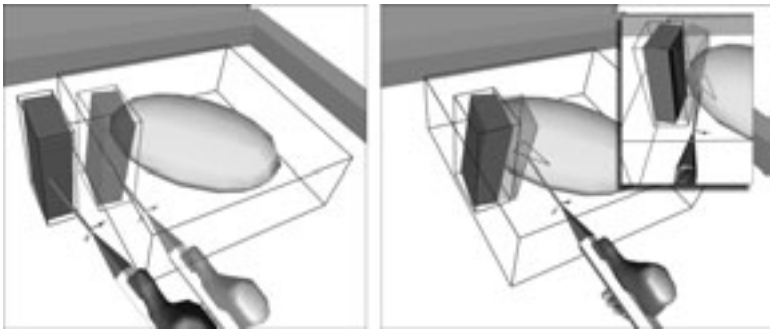


Figure 4.5: User manipulation of an object before (left) and during collision (right). Objects cannot move through each other. The colliding triangles are highlighted (right).

Collision detection and handling is important for realistic object behaviour during its motion or manipulation by a user. We have to prevent objects from moving through each other. Detecting object collisions and the collision handling helps to create an illusion that virtual objects have a substance, see Figure 4.5.

Based on the experiences with approaches mentioned above, we have implemented the following object collision schemes: *collision of stylus-object, ray-object and object-object.* The stylus and ray intersections are supported by Iris Performer [Rohlf & Helman, 1994]. Iris Performer provides axis-aligned bounding boxes, bounding spheres and cylinders. Into the RWB-Object class we have implemented a complete support for oriented bounding boxes, which much better fit around objects than axis-aligned boxes. Certainly, it is not too computationally intensive to determine whether a point is inside a bounding volume or if a ray intersects these types of bounding volumes. The point and ray casting queries on bounding volumes of objects in a virtual world are the basis of our object selection approach.

Collisions between objects have been implemented into the RWB Library. Bounding volume collisions between objects are evaluated first. When we detect any overlapping bounding boxes or spheres the system performs also a precise polygonal collision check. This part of the collision detection can put serious limitations on performance, especially if the triangle collision is not optimized and all triangles of one object are tested against triangles of the other object. Therefore we have implemented certain optimizations in order to check only a small amount of triangles which could potentially collide.

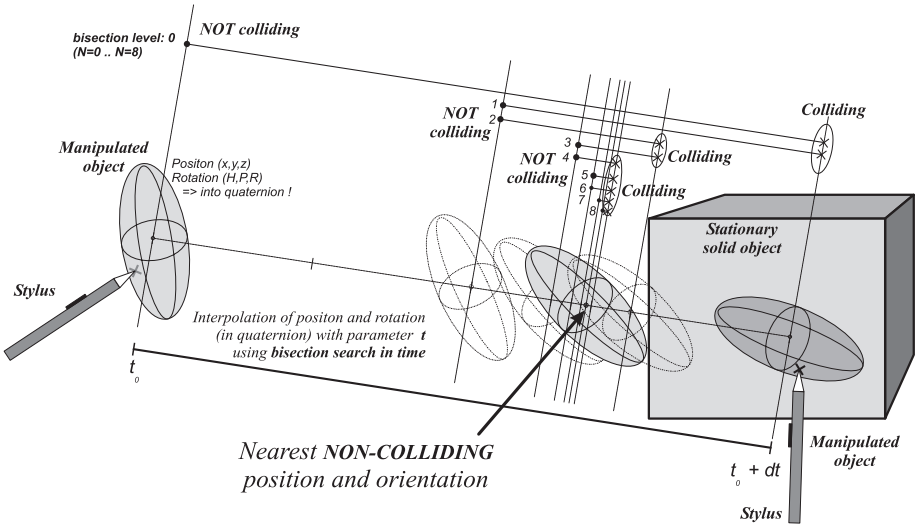


Figure 4.6: RWB-Object collider routine finds a contact configuration of the manipulated object with the stationary object.

After detection of the collision we have to respond to it in an appropriate way. In case of an object manipulation by the user we have to prevent the collision and find the contact configuration (position and orientation) of the objects, see Figure 4.6.

Here we see the manipulated object that is not colliding at time t_0. In the next time step $t_0 + \Delta t$ we have detected a collision. Since we don't know the exact motion of the manipulated object we cannot determine analytically the time of collision and its place. It becomes even more complicated when we imagine more complex objects. We have to solve this problem iteratively. We use the *bisection search in time.* As Figure 4.6 shows, we are stepping to $t_0 + \frac{1}{2}\Delta t$. If the manipulated object penetrates the other object at $t_0 + \frac{1}{2}\Delta t$ time then we would try stepping to $t_0 + \frac{1}{4}\Delta t$ but if it wasn't penetrating we would step from $t_0 + \frac{1}{2}\Delta t$ to $t_0 + \frac{3}{4}\Delta t$. In the same fashion we continue until the objects are close enough. From our collision experiments we have found that 8 levels of bisection are sufficient. We have considered the speed of the user's hand motion and the maximal length of motion on the Workbench, which is about 180cm. The tracker daemon measures the stylus position every 20 miliseconds. In such a short time the user can move his/her hand about 30-40cm at most. Thus, when we use 8 levels of bisection, it gives us a collision-resolution of $\frac{40}{256} = 0,15$ cm. As we have to interpolate the position and rotation over time, we have decided to use quaternions to encode the vector of the object's orientation (H,P,R). It results in a much more logical orientation over interpolation time.

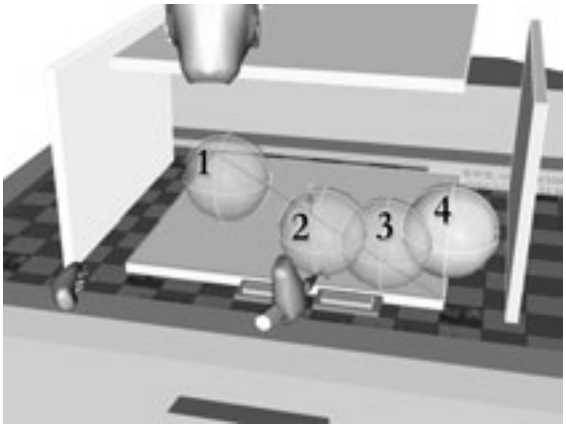


Figure 4.7: Dynamic simulation of a bouncing ball. The user can catch the ball and throw it against the walls. Elastic collisions are simulated. The triangle colliding with the bounding sphere is highlighted.

When dynamic object behaviour is simulated (like in Figure 4.7), the object-collider routine has to detect the colliding triangles of objects, and also the collision plane and its normal vector have to be determined.

Object Constraints

Similar to the real world, object manipulation must be constrained by the surrounding environment and by the properties of the object itself. Imagine a heavy object that is resting on the ground and has to pushed to another location, or a key that has to be inserted into a lock. When creating virtual environments that should mimic reality it is important to incorporate constraints, otherwise it will appear very unrealistic.

We can distinguish between geometric constraints and dynamic constraints. A proper simulation of motion constraints has to be physically-based [Barzel & Barr, 1988]. When we consider a scene consisting of rigid bodies, which act in accordance with the rules of physics, we could implement the constrained object behaviour via a set of constraint forces. Generally, the contact behaviour of objects, which appears for example during virtual assembly simulations, is very difficult to simulate with interactive frame rates. Certainly, for the purposes of Virtual Reality we use a simplification of the underlying physics, as we have done in our testing object-assembly application of the Spring Manipulation Tools, see Section 4.2.1. It seems that users have difficulties to freely manipulate objects in three dimensions without any constraining or supporting objects.

Geometric constraints have a close relation with object collisions. When we try to avoid object collisions in fact we apply constraints. However, the collision checking is often computationally very intensive. In some cases, we can apply constraints on object manipulation in an elegant and efficient way without an explicit need to perform the collision checking. Geometric constraints are very well established phenomena in world of 3D geometric modeling programs (3D Studio Max, Maya, AutoCAD, etc.).

A simple approach of constrained manipulation for VR has been presented in UNC-CHIMP (an immersive modeling program) [Mine, 1996]. The developers implemented support for movement along a line or in a plane, rotation about any vector, full rotation (HPR) about the object's center, and uniform scaling about the object's center. After selection of a suitable manipulation technique (translation, rotation, scaling) the user has to specify the axis of the manipulation, and then the magnitude of the manipulation by moving of the interaction device. This way 1D or 2D constrained manipulation can be easily implemented, see Figure 4.8.

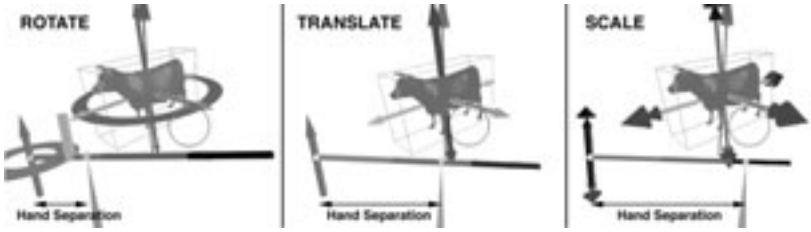


Figure 4.8: One-dimension constrained object manipulation [UNC-CHIMP].

Another approach of constrained object manipulation [Smith & Stürzlinger, 2001] is based on the semantics of objects in the virtual world. Each type of object incorporates its own semantic rules and relations to other objects in the scene. In an example of a room with a table, a chair, a desk lamp, a computer, and a keyboard, the scene graph with constraint relations is shown. The chair is placed on the floor and is attached to the table. The computer with the keyboard, and the desk lamp can move only on the surface of the table. In addition, the desk lamp has to be on the right side of the computer. Such rules can be incorporated into the scene model. During manipulation of objects in the scene the satisfaction of the constraints is evaluated.

Our approach of object constraints was developed to support the process of data exploration in an intuitive manner. We needed to navigate in a 3D visualized scene of data, precisely manipulate the visualization tools, operate the 3D widgets, etc. For these purposes we have implemented a geometric constraint scheme. Object manipulation (translation, rotation, and scaling) is performed by applying homogeneous transformations. Matrices can effectively encode the desired object transformation from one position to another. For each RWB-Object in the scene we can specify a constraint mask, see Figure 4.9.

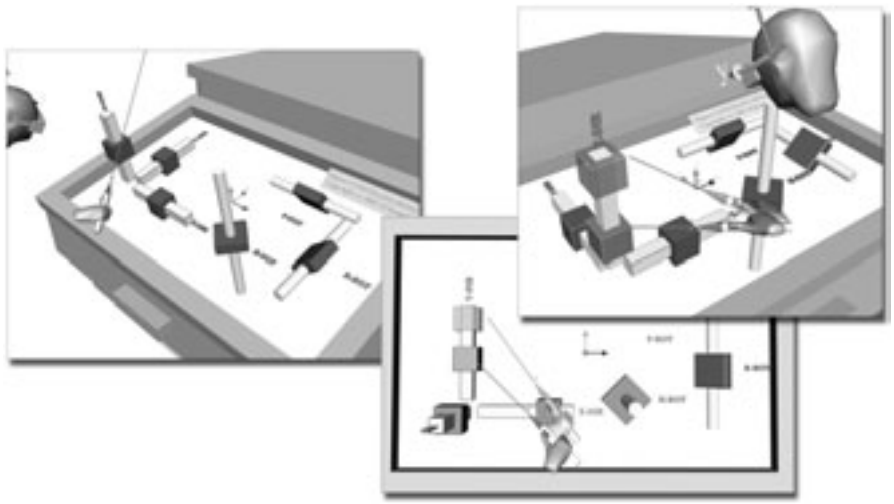


Figure 4.9: Object constraints: translation constraints (X-dir, Y-dir, Z-dir) and rotation constraints (HPR - heading, pitch, roll)

When the transformation matrices are orthogonal (only uniform scaling can be present in the matrices), then we can extract from these matrices the translation and rotation vectors, (x, y, z) and (h, p, r). We can then apply the constraint mask of the manipulated object to this vector. After this constraining action the resulting vector is converted back into the transformation matrix, which finally moves with the objects only in the constrained directions.

With this very simple and fast method we can constrain objects to any plane, line, axis direction, or to permit rotation in heading, pitch, or roll.

In the following examples (Figures 4.10, 4.11) we show some applications of our object-constraint approach.

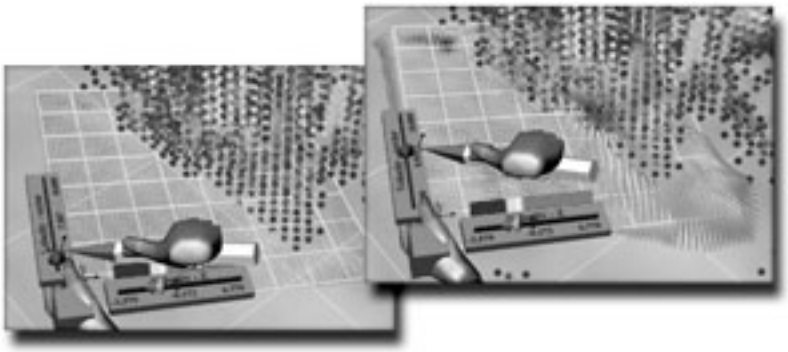


Figure 4.10: RWB-Widget-slider: the slider element is constrained to move along the slider line. The user changes the scaling factor of the vector-data-slicer.

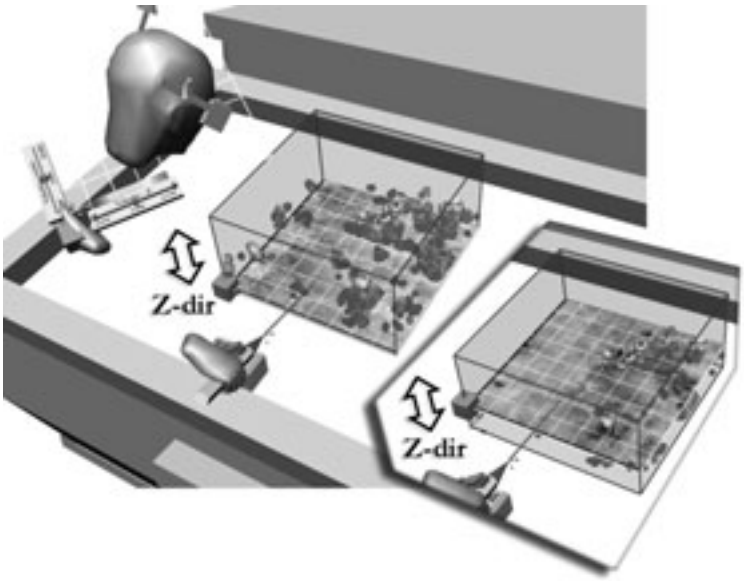


Figure 4.11: VRX-data-slicer can move only vertically. More about VRX visualization techniques can be found in Section 5.2.

4.2 Force Feedback and Spring-Based Tools

In user interaction with virtual worlds, consistent and realistic behavior of objects is very important. We want objects to respond in a natural and predictable way to our actions. But usually virtual objects are weightless and unsubstantial, and they move without friction of inertia; this leads to altogether 'unphysical' behavior and unpredictable responses, especially in semi-immersive environments such as the Responsive Workbench, where real and virtual worlds coexist, and should follow the same natural laws. Absence of weight and substance may sometimes be desirable while inspecting an object or flying through an environment. But especially in manipulation tasks, mechanically realistic behavior can help to achieve consistency and predictability.

Passive Haptic Feedback

Virtual Reality provides mainly computer-generated visual stimuli, sometimes supplemented with audio or trying to stimulate other senses also. In the real world we make a daily use of passive tactile/haptic feedback provided by real objects everywhere around us. It is not an easy task to simulate a real experience of touching objects in virtual environments. One of the most disconcertingly unnatural properties of VEs is the ability of users to move through the virtual objects. This reminds users that the perceived environment is not real, reducing their sense of presence.

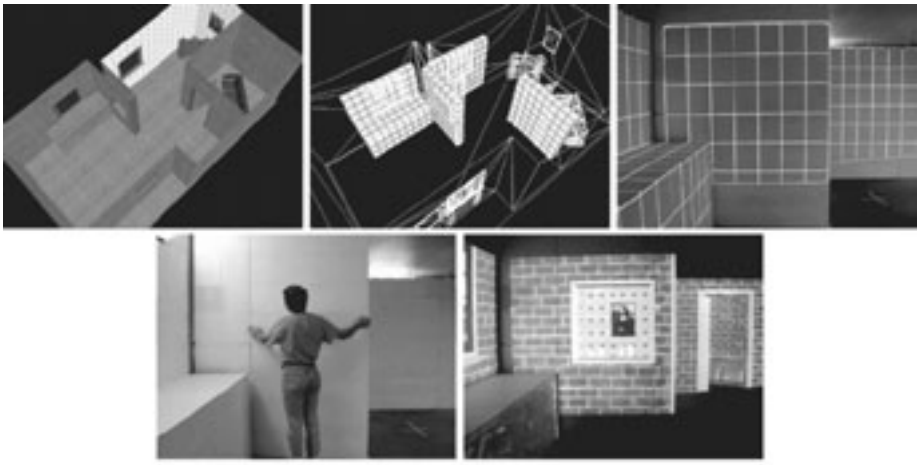


Figure 4.12: Passive haptic feedback environment [UNC: Being-there project]

Passive haptics incorporates passive physical (real) objects into VEs to physically simulate the virtual objects. This way the user can experience touching of virtual objects. The most successful real-world simulators, such as flight, ship and driving simulators, physically replicate anything that the user might touch, from steering to dials and buttons.

In these cases it is feasible because the displayed VE is outside the vehicle. Sometimes even whole cars (or at least detailed mockups) are installed for these purposes. However, in many other applications it is not feasible at all, technically and economically. Passive haptics can be used then at least partially, to improve the presence in the VE. An example with a walkthrough is shown in Figures 4.12 and 4.13. The passive haptic feedback significantly affects "the sense of being present in the virtual environment" [Insko, 2001].

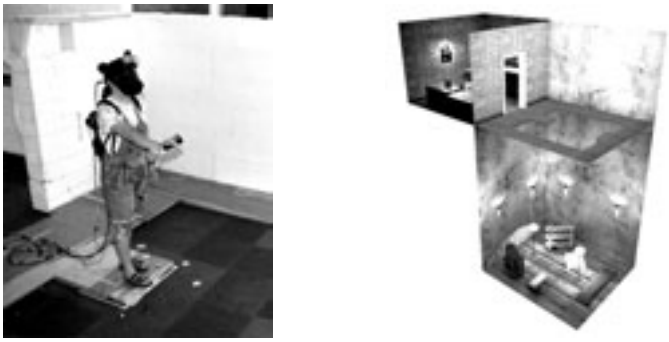


Figure 4.13: Passive haptic feedback improves the presence in the VE [UNC]

In the case of the Virtual Workbench, we want to build upon the laboratory table metaphor, with instruments resting on the table and the laboratory experiment presented on the table. The Workbench table has a solid wooden framework, thus the user can physically touch the table and glass display screen. Users usually prefer direct interaction with objects and widgets of the 3D GUI instead of virtual laser pointer (ray-casting) and they only choose the ray-casting if the objects are out of reach. This can happen with distant objects or when an object is relatively small and hanging "somewhere above the table". What makes it difficult to select these hanging objects is the missing feedback of physical contact with the objects. Our sense of touch is not stimulated.

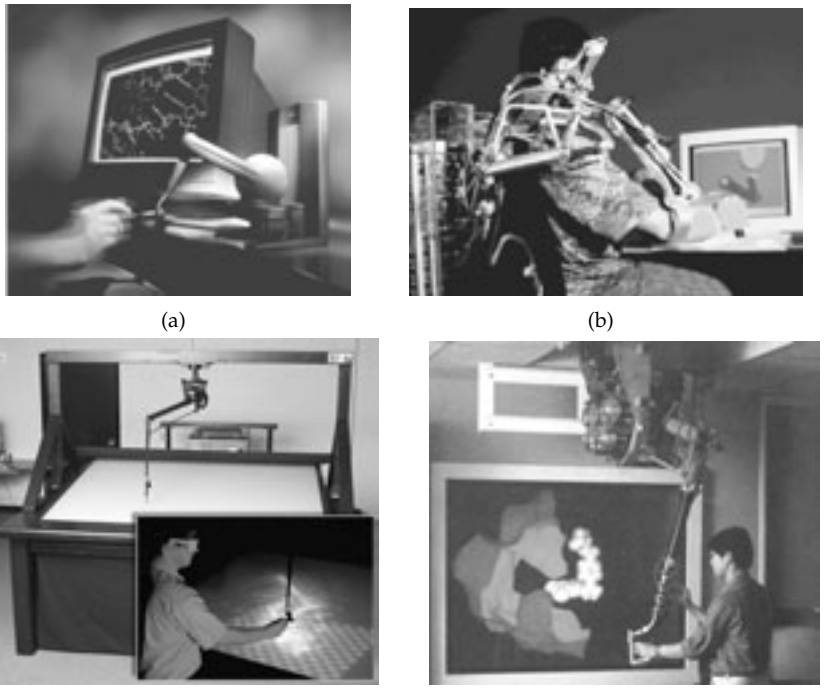
Figure 4.14: RWB - passive haptic feedback provided by the Plexipad

In our concept we make use of the physical (passive haptic) feedback which is provided by the glass screen and the Plexipad. On both these surfaces we place 3D widget containers with buttons and sliders, which are easy to reach

and operate, see Figure 4.14 Especially, the Plexipad can provide much more than this. As it is a transparent acrylic plate, we can project any image on the plane of this plate. It can perform the task of a visualization or probing tool with controls mounted on it, see Section 5.1.4.

Active Haptic Force Feedback

A way to provide active mechanical responses from a virtual world is through haptic devices [Massie & Salisbury, 1994; Burdea, 1996; Ruspini *et al.*, 1997; Grant *et al.*, 1998]. Haptic devices usually give force feedback through an intermediary device with a limited range, see Figure 4.15. Force feedback can be very useful, especially when some mechanical device is used for interaction with the VE, such as mechanical grasping devices (pantographs), or surgical instruments. However the technology for free-field haptic interaction, where objects may be touched anywhere in space, is still immature. Therefore, we have searched for an alternative to the use of force-based haptic interfaces.



(a) (b) (c) (d)

Figure 4.15: (a) Molecular interaction with a Phantom-like haptic device; (b) Pneumatic robot haptic arm; (c) SCI Haptic Workbench; (d) UNC molecular docking installation.

We will look for a limited set of physical properties and laws of behavior to make user interaction more predictable and intuitive. We will introduce the

concepts of force, inertia, gravity, contact, surface friction, and damping into the virtual world. Thus, we will provide a visual interface to replace direct force input; we call it a *visual force feedback.*

4.2.1 Spring-Based Manipulation Tools

In this section we present new tools for user interaction with virtual worlds, to bring more natural behavior into the manipulation of objects in virtual environments. We present some principles of physically realistic behavior of virtual objects and a set of user input techniques suitable for semi-immersive VR devices such as the Responsive Workbench. We present a spring-based visual force-feedback method and we provide a visual interface as an emulation of direct force input.

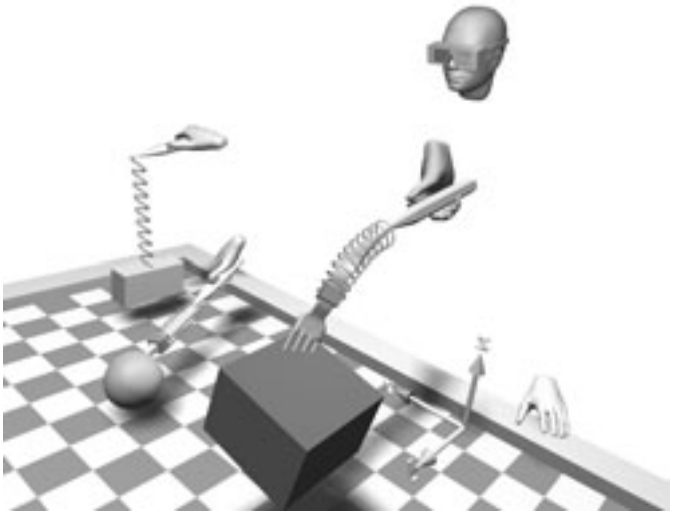


Figure 4.16: Illustration of spring-based tools

We do this by the use of spring-based tools attached to objects assisting the manipulation, based on the following assumptions:

- A linear relation of force with spring compression / extension is intuitively understood and visualized by the spiraling shape of a spring. Thus, even without exerting real force, a user has an intuitive notion of transforming a change of spring length to a force.
- Bending and torsion of a shaft is used to show forces and torques exerted on virtual objects
- The relation between object mass and size is also intuitive. Specific mass of objects should be user-specified. A massless world can always be created by setting all specific masses to zero.

- The RWB provides a natural ground plane for objects at rest.
- Stability is introduced by friction and damping, reducing excessive effects of input actions on objects, and reducing undesired oscillations.
- Physical contact of objects is intuitively equivalent with geometric intersection, and sound can be used to provide contact feedback.

We introduce a set of **spring-based tools** for providing the basic manipulation tasks (see Figure 4.16). We hypothesize that dynamics will provide intuitively consistent and predictable behavior of the objects in the virtual world, and that interactive manipulation will therefore be easier and more natural.

- A **Spring** is attached to the center of an object. It assists linear motions (translation). The tool has 1 DOF (degree of freedom), the length of the spring, and controls 3 DOF (x,y,z) of an object.
- A **Spring-fork** is attached to an object and defines a contact point for transfer of forces and moments to the object. It assists translations and rotations. The tool has 3 DOF (extension, bending, torsion), and controls 6 DOF (x,y,z + h,p,r) of an object.
- A **Spring-probe** can be used for probing the material stiffness of an object or pushing an object. The tool has 1 DOF (length) and can control 3 DOF (x,y,z) or 1 DOF (pressure) of an object.

Why spring-based tools?

In VR it is just as easy to manipulate large and heavy objects as small ones. This may sometimes be an advantage, but in general it is not according to our expectations. Therefore, we will try to give the user a sense of weight or mass of objects. We propose to use the spring-tools as a link between the user's hand and a manipulated object. In this way, we can obtain a natural visual feedback during the manipulation. When the user lifts a heavy object, the spring will extend proportionally to the object's weight. Also, acceleration and deceleration of the motion will affect the visible length of the spring.

The fork metaphor seems to be very intuitive. For object selection the fork has to be inserted into an object. The user can fix the position and the orientation of the fork inside the object. The spring part of the fork gives a visual dynamic feedback during the manipulation of the object. It is also intuitive that rotating the fork should rotate the object. The user controls one end of the fork and the other end is influenced by the object. The fork can bend, extend (compress) or twist according to the laws of mechanics.

We have demonstrated the usage of the spring-tools with a simple object assembly task, which is easy in a real world but hard to simulate in a virtual environment; see Figures 4.17, 4.22 and 4.31.

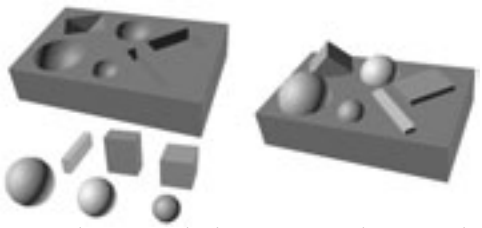


Figure 4.17: Manipulation task that was used to test the spring tools.

Related Work

We first review some relevant related work on aspects of the dynamic and the spring interaction techniques in VR. The basic interaction techniques have been already discussed in Section 4.1.

The problems of mechanics and dynamics have been studied extensively from many different points of view. It is outside the scope of this thesis to review this field. We do not have the intention to implement a fully realistic dynamic manipulation system, but rather a limited and simplified set of basic physics to assist the manipulation of objects in VE.

The springs are very well established phenomena in the world of physics and graphics as well. Springs were used also by others for manipulation of objects on the Responsive Workbench [Fröhlich *et al.*, 2000]. The authors demonstrate the use of simple 1 DOF springs in set of multi-spring configurations to manipulate objects, giving visual force feedback. This interaction approach allows multiple hands and multiple users to manipulate the same object. To simulate the spring behaviour they use the Coriolis physical simulation package. The simulation update of 5 Hz seems to be really difficult to use for interactive visual force feedback, which they claim to achieve.

We have introduced the spring manipulation tools in [Koutek & Post, 2000; Koutek & Post, 2001c], where we have demonstrated the use of spring for manipulation of objects in virtual environments in simple assembly tasks. Our work began with a simple spring manipulation tool [Koutek & Post, 2000], which could only translate objects giving a visual feedback on inertial effects. The spring-fork [Koutek & Post, 2001c] needs just one hand with the stylus to control position and rotation of attached object, and it provides a clear visual force feedback. When we look at frame rates (> 24Hz), our simulation of spring deformation is not a bottleneck in the visualization and interaction pipeline.

Our approach has a set of simple manipulation tools which reflect dynamic object behavior during the manipulation. Our first interest was in single-handed manipulations.

We have extended the spring metaphor for manipulation of particles in steering of Molecular Dynamics simulation [Koutek *et al.*, 2002].

4.2.2 Dynamics on the Responsive Workbench

As discussed before, we have introduced the concepts of force, inertia, gravity, contact and surface friction to provide a virtual world in which objects behave more naturally. Objects behave according to physical laws and also the interaction is physics-based.

We have applied the relevant laws of mechanics. In case of object collisions we had to deal with principles of conservation of momentum and of energy, and reversible conversion between kinetic and potential energy.

Method

A specific mass is assigned to each object. From the volume of an object its mass is calculated. To make objects move, forces are needed. In our case, *the gravity force* and *the force exerted by the user* are acting. Counter-acting forces are *the friction force* between the object and the surface and *the air resistance force*. For stability *a damping force* is applied. We use a simplified implementation of the physics laws in our virtual world. We will ignore the *air-resistance.* We will only use the static friction μ, when user drags an object across a surface. At low speeds μ is constant.

When the forces act on the object at a certain distance from the center of mass, it induces a moment and a torsion. The final rotational motion depends also on the moment of inertia of the object which corresponds to the axis of rotation. For simplicity, we will consider now that the whole mass of the object is concentrated at its center of mass and therefore the moment of inertia around axis O will be $I_O = r_O^2 m$, where r_O is the perpendicular distance of the center of mass from the axis O.

To make the simulation more realistic, it is possible to include a precise calculation of the moment of inertia for a given object and a certain axis of rotation. Here is an overview of the laws of mechanics as discussed above:

Force law: $\mathbf{f}_a = m\mathbf{a} = m\frac{d\mathbf{v}}{dt} = m\frac{d^2\mathbf{x}}{dt^2}$

Gravity law: $\mathbf{f}_g = m\mathbf{g}$

Friction force: $\mathbf{f}_r = \mu\mathbf{N} = \mu.m\mathbf{g}$

Spring force: $\mathbf{f}_s = -k\mathbf{x}$

Damping force: $\mathbf{f}_d = -c.\frac{d\mathbf{x}}{dt}$

Linear Momentum:
$$\mathbf{F} = m\mathbf{a} = m\dot{\mathbf{v}} = \sum_{i=1}^{n} \mathbf{F}_i$$

Angular Momentum:
$$\mathbf{M} = I\boldsymbol{\epsilon} = I\dot{\boldsymbol{\omega}} = \sum_{i=1}^{n} (\mathbf{r}_i \times \mathbf{F}_i)$$

Where: m=mass, $\mathbf{f}$=force, $\mathbf{x}$=position, $\mathbf{v}$=velocity, $\mathbf{a}$=acceleration, $\mathbf{g}$= gravity acceleration, μ= static friction coefficient, k= spring constant and c= damping factor, $\boldsymbol{\omega}$=angular velocity, $\boldsymbol{\epsilon}$=angular acceleration, $\mathbf{r}_i$=force position, I=moment of inertia.

During manipulation, a spring-tool is attached to an object. Each spring-tool *(spring, spring-fork, spring-probe)* has to be calibrated with the manipulated

object. This is done automatically using the weight and the dimensions of the object. The calibration procedure consists of calculating spring and damping constants and choosing a suitable size of the spring-tool. For practical reasons, a rigid, inflexible and large spring-tool is used for heavy and large objects, and a flexible and small spring-tool for light objects, see Figure 4.18.

$$\begin{aligned} \text{Spring constant:} \quad k &= -\frac{m.g}{x_m} = \frac{m.g}{0.2x_0} \\ \text{where} \quad mg &= -kx_m; \quad x_m = -0.2x_0 \end{aligned}$$

x_0 - initial length of the spring; x_m - extension with load m

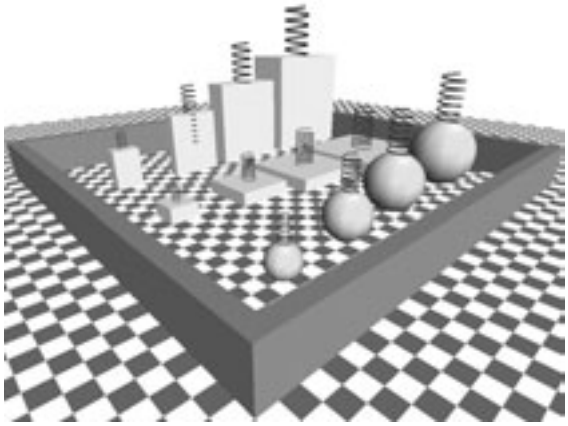


Figure 4.18: Various objects and different springs

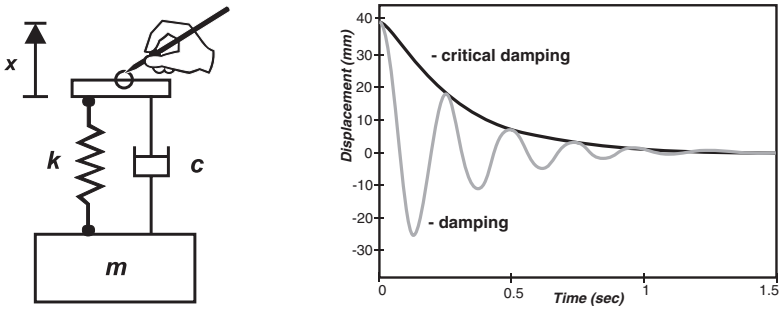
Spring Damper System

The main component of spring-tools is the spring-damper. To reduce oscillations during manipulation, when an object is attached to a spring-tool, we have implemented a *spring mass-damper system* for each DOF, Figure 4.19(a). The force $\mathbf{f}(t)$ acting on an object as a function of time t is:

$$\mathbf{f}(t) = m\ddot{\mathbf{x}}(t) + c\dot{\mathbf{x}}(t) + k\mathbf{x}(t) \tag{4.1}$$

Here m is the mass of the object, t is the time, $\mathbf{x}(t)$ is the extension / compression of the spring, $\mathbf{f}(t)$ is the force acting on the object only in $\mathbf{x}(t)$ direction, k is the spring constant and c is the damping factor, $\mathbf{g}$ is the gravitation acceleration.

If $\mathbf{x}_s$ is the static extension of the spring and $\mathbf{x}_d$ is the dynamic change of extension during the motion, we can simplify the spring-damper formula 4.1 by assuming: $\mathbf{x} = \mathbf{x}_s + \mathbf{x}_d$ and $k\mathbf{x}_s = m\mathbf{g}$.



(a) (b)

Figure 4.19: (a) Spring-damper system; (b) damping comparison.

Applying this assumption we will get the following system of differential equations with initial conditions for the dynamic spring-damper motion $\mathbf{x}_d$:

$$m\ddot{\mathbf{x}}_d + c\dot{\mathbf{x}}_d + k\mathbf{x}_d = 0; \qquad \mathbf{x}(0) = \mathbf{x}_0, \quad \dot{\mathbf{x}}(0) = \mathbf{v}_0$$

As the spring is a natural mechanical oscillator it would not be very suitable for object manipulation. With our virtual environment we can create an artificial (virtual) behaviour of the spring, which will fit into our interaction scheme. To avoid the unwanted oscillations of the spring-damper we use a critical damping solution:

$$\mathbf{x}_d(t) = (A.t + B)e^{-\alpha t}; \quad with\ A = \mathbf{v}_0 + \mathbf{x}_0\alpha;\ B = \mathbf{x}_0$$
$$where \quad \alpha = \tfrac{c}{2m}; \quad \omega_0 = \sqrt{k/m}$$

For critical damping: $\alpha = \omega_0$ thus $c = 2\sqrt{mk}$. The damping factor plays a very important role. Its value is derived in a similar way as the spring constant, so that every object has its spring and damper. *Damping* refers to an energy dissipation mechanism, either intentional or parasitic, such as air friction or structural damping. The *damper* is the energy dissipating element of the system. The *damping factor* measures the ability to damp the motion of the mass. The spring-damper is calibrated to hold the attached object with a certain extension. In Figure 4.19(b), the effect of the damping factor can be observed. It is clear that the critical damping without any oscillation converges to zero but in fact it takes an "infinite" time to reach zero. For practical reasons when the spring-damper simulation reaches a threshold, it stops.

The initial spring-damper conditions have to be changed as the user moves the spring-tool. Therefore, we have to re-initialize $\mathbf{x}_0$ and $\mathbf{v}_0$ at each simulation time step. The effect is that the spring-tool is always trying to restore its statically balanced configuration, while being unbalanced by the user interaction and the object inertia.

4.2.3 Spring Manipulation Techniques

An object can be manipulated using a set of techniques as described below. A spiral spring is used as a handle, and the object will show its mass and inertia by its behaviour according to the laws of dynamics.

The actions are performed by selecting an object by moving the stylus towards it. As the stylus is moved closer to the object, a spring will be displayed between the object and the stylus tip position. The spring can be extended or compressed by keeping the pen button depressed while moving the pen towards or away from the object.

Virtual forces are thus applied by the user to objects through the virtual springs, which act as displacement-to-force transducers. The user will see the extension or compression of a spring, and forces are inferred by the linear relation of displacement and force. The forces may be exerted on an object in an arbitrary direction. They are assumed to work on the object's center of mass, and thus the forces will induce only linear motion; no rotational motion, as no torques are induced. Although friction on the ground plane can induce torques and rolling motions, these effects are ignored in the current version of the system.

The exerted forces are decomposed into lifting forces (perpendicular to the ground plane), and dragging forces (parallel to the ground plane). To ensure easy and stable manipulation, both linear and pendulum oscillations are damped critically, which means that motion will essentially stop after one cycle. For clarity, the objects are assumed here to rest on the ground plane, although other horizontal planes will have the same effects.

Fig.4.20 shows the three cases: *lifting*, *pulling* and *pushing*. The user exerts a force F_u which is unknown. From the change of extension of the spring $\mathbf{x}_1$ we can obtain the spring force $\mathbf{F}_s$ which makes the object attached to the spring accelerate and move.

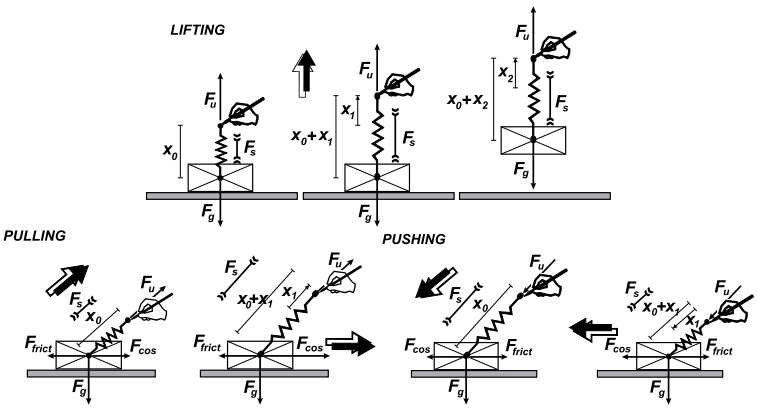


Figure 4.20: Spring manipulation cases

The set of dynamic manipulation techniques is defined as follows:

- **Lift:** pull the spring upward, until the spring force exceeds the object's weight, and the object gets an upward acceleration, counter-acted by the decrease on the force caused by the shortening of the spring length. Oscillation is damped.
- **Drop:** the object is released from the spring connection and falls down until it touches the ground plane; comes to rest immediately (no bouncing).
- **Pull:** pull the spring in a horizontal direction away from the object until the force exceeds the static friction of the object on the ground surface. The object gets a horizontal acceleration and slides over the surface, counter-acted by friction force, and the decreasing force from the shortening of the spring.
- **Push:** the same as pull, but the spring is compressed toward the object.
- **Throw:** the object receives the initial velocity from the user's hand and is thrown in the given direction, the spring disappears. The trajectory of the object in the air is then only influenced by the gravitation force and the object is falling to the ground.
- **Catch:** stop the object motion by blocking its motion path. The spring between the user's hand and the object is compressed and the object decelerates and stops moving. Oscillations are again damped.
- **Swing:** a combination of lift-pull-drop. The object behaves as a pendulum on the spring; the swinging motion is damped, the object is moved through in the air, and is dropped at the chosen position.

The difference between dropping and throwing is made by analysis of hand motion data from the tracker. With dropping, the pen button is released while the user's hand is at rest, for throwing, the user releases the button while the hand is moving. Catching flying objects can be done by positioning the hand to block the motion path and pressing the button.

Wherever spring actions are performed (too) far from the user's body, a *'fishing rod' technique* can be used to attach and move the spring and the object, see also Section 4.2.5.

Collisions and Constraints

For natural behavior in virtual environments, the collisions and constraints are essential. Of course, we cannot stop the user's hand to move through objects. But we can disable user manipulation of an object through another object. We can make use of the advantages of the spring manipulation.

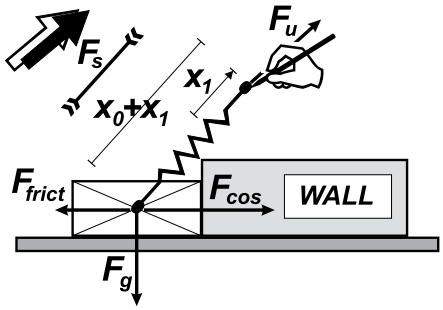
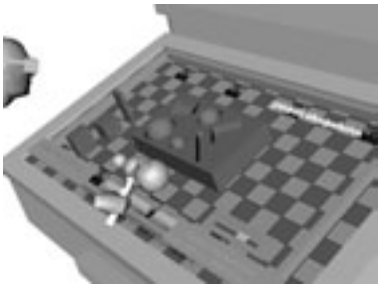


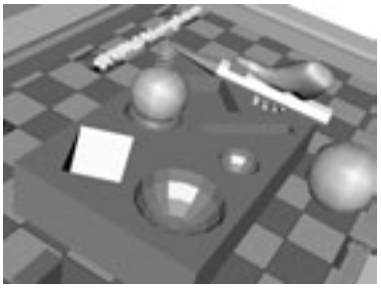
Figure 4.21: Collision with the wall

The manipulated object will stay at the place of collision with the other object (see collision with a wall, Figure 4.21) while the user is still trying to pull. In this case, the pulling will only affect the length of the spring, increasing its tension, and no motion is induced.

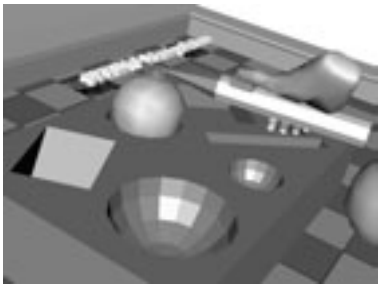
The basic constraint is the ground plane. No object can get through it. The ground can always produce enough reaction force to support any object. Calculation of collisions with the ground is the basic attribute of our mini-world. We also handle collisions between objects inside the world (spheres, boxes). Elastic collisions can be simulated for this. We use the law of momentum conservation and the law of energy conservation.



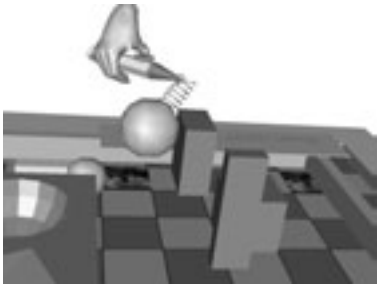
(a) Object selection with the spring



(b) Object attached to the spring



(c) Object assembly using the spring



(d) Visualization of object collision

Figure 4.22: Spring tool demonstration (see also the Color Section)

4.2.4 Spring-fork: A Flexible Manipulation Tool

The spring-fork is attached to an object and defines a contact point for transfer of forces and moments to the object. It assists translations and rotations. The tool has 3 DOF (extension, bending, torsion) and controls 6 DOF (x,y,z + h,p,r) of an object. Each spring-damper in each DOF is calibrated to hold the attached object with a certain extension. In case of the spring-fork (see Figure 4.23) there are three spring-dampers, one for each DOF: bend, stretch, torsion.

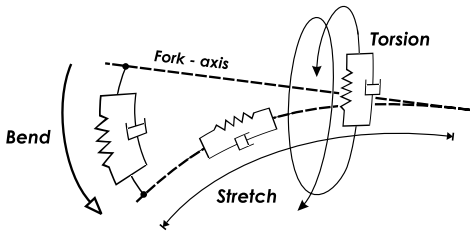


Figure 4.23: Spring-damper systems for each deformation parameter

Spring-fork Model

To implement the spring-fork manipulation, we first need to create a model of the spring-fork that would be scalable and deformable with bending, stretching (extension) and torsion deformations. The fork consists of a *rigid part* (the fork end) and a *flexible part* (the spring part), see Figure 4.24.

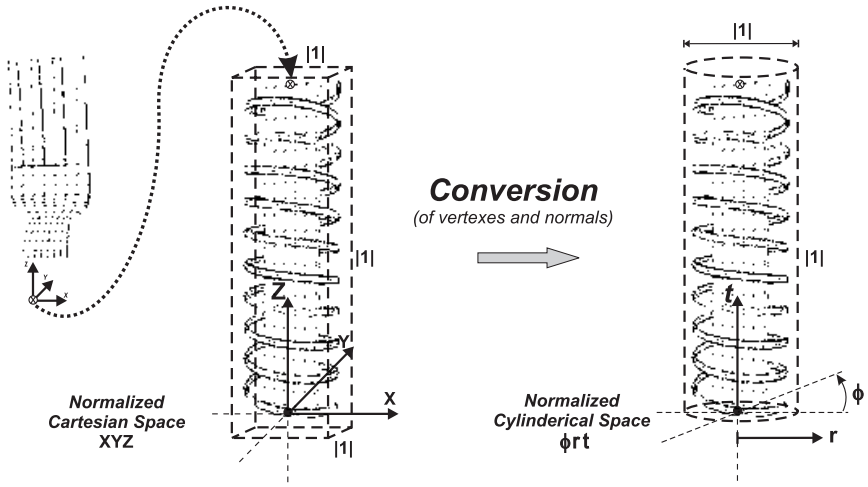


Figure 4.24: Space conversions of the fork's vertices and surface normals

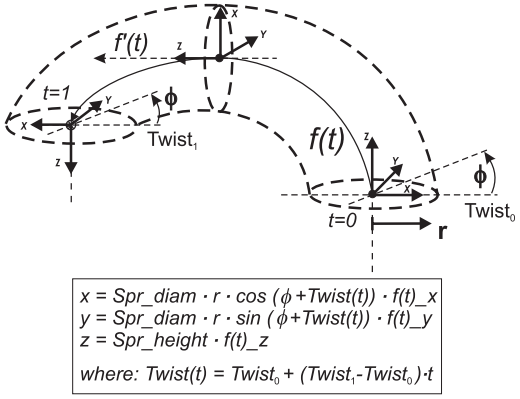


Figure 4.25: Reconstruction of the XYZ coordinates of the deformed fork

The 3D model of the fork is converted to a normalized cylindrical coordinate space, which is used to perform the fork deformation. All vertices and normal vectors are converted once and stored in memory as the initial and the normalized form of the fork. For a given scale of the spring-fork *(Spr_diam, Spr_height)* and a given deformation *(bend, stretch, torsion)* we have to reconstruct the final XYZ coordinates (Figure 4.25).

The torsion parameter of the spring-fork is the difference between $twist_0$ and $twist_1$. The bend and the stretch parameters are built in a parametric function $f(t)$. From Figure 4.25, it is clear how the LCS (local coordinate system) rotates along the function $f(t)$. So for each position at $f(t)$ we need to know the tangent vector, thus $f'(t)$ which corresponds to the Z-direction of the LCS. The fork-end is attached at the end of the spring and it is aligned with the last LCS.

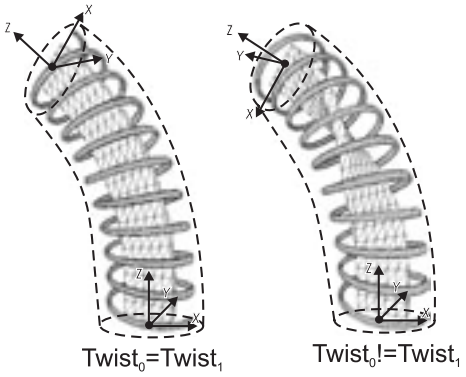
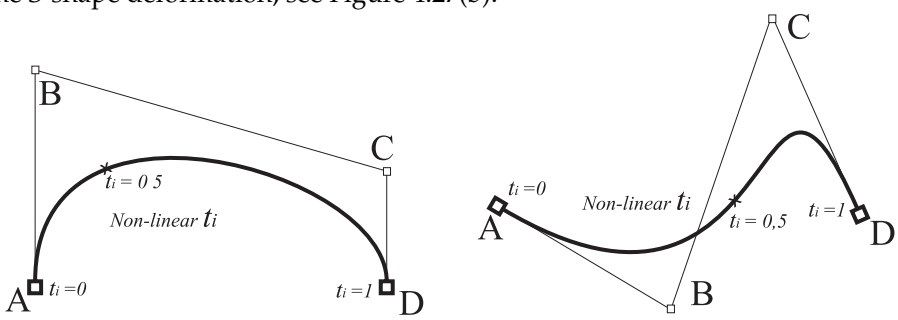


Figure 4.26: Deformation of the spring-part of the tool

Figure 4.26 shows deformations of the spring-part of the fork. The first example is without torsion and the second shows the both bending and torsion.

The deformation function $f(t)$ needs explanation. The problem of object or beam bending is not trivial. Bending depends on the profile along the length the object, on the material structure, on the type of load, and on other parameters. In general it is a complex problem. A full general solution of the mechanical problem would be too time consuming, so we will use a simplified solution that will satisfy our needs.

We will assume that the load on the fork is concentrated. We designed an *arc-length parameterized Bézier spline function for bending* (Figure 4.28), and allowing the S-shape deformation, see Figure 4.27(b).



(a) Normal Bézier spline segment (b) S-shaped Bézier spline segment

Figure 4.27: Bézier curve is given by end points A,D and control points B,C.

A cubic Bézier curve $Q(t)$ is defined by two end points A,D and two control points B,C:

$$Q(t) = AT_1(t) + BT_2(t) + CT_3(t) + DT_4(t) \qquad where \quad \mathrm{t} \in < 0, 1 > \tag{4.2}$$

The Bézier curve contains Bernstein polynomials $T_{1,2,3,4}$:

$$T_1(t) = (1-t)^3 \qquad T_2(t) = 3t(1-t)^2 \qquad T_3(t) = 3t^2(1-t) \qquad T_4(t) = t^3$$

$$where \qquad \sum_{\mathrm{i}=1}^{4} \mathrm{T_i(t)} = 1$$

The Bézier curve parametrization can be written in the following form:

$$\begin{aligned} x: \quad \phi(t) &= A_xT_1(t) + B_xT_2(t) + C_xT_3(t) + D_xT_4(t) \\ y: \quad \psi(t) &= A_yT_1(t) + B_yT_2(t) + C_yT_3(t) + D_yT_4(t) \\ z: \quad \theta(t) &= A_zT_1(t) + B_zT_2(t) + C_zT_3(t) + D_zT_4(t) \qquad with \quad \mathrm{t} \in < \alpha, \beta > \end{aligned} \tag{4.3}$$

The arc of the Bézier spline can be computed with the following formula:

$$S(c) = \int_{\alpha}^{\beta} \sqrt{(\phi'(t))^2 + (\psi'(t))^2 + (\theta'(t))^2}dt \tag{4.4}$$

A Bézier arc $S(c)$ defined in this way is in fact measuring the length of the spline. When $\alpha = 0; \beta = 1$ then $S(c)$ is equal to the total arc length L. $S(c)$ implements a linear metrics on the arc of the Bézier spline. From Figure 4.27 is clear that the original parameter t is not linearized ($t = 0.5$ is not in the middle of the arc). As we use Bézier curves for geometric deformation of objects we must have linearized (arc-length) parameterization of this spline. We don't use the original abstract non-linear parameter t. Instead, we use arc-length parameter l, see Figure 4.28.

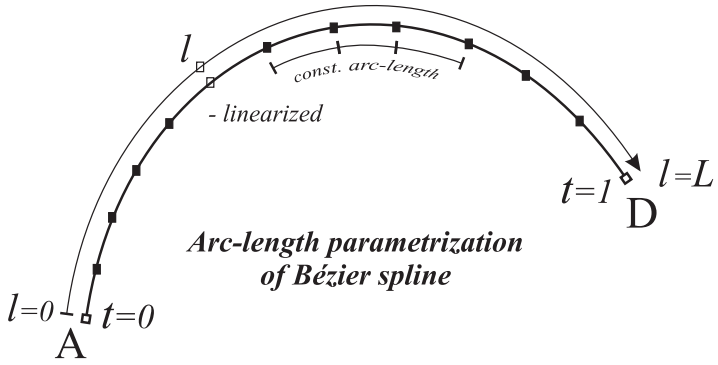


Figure 4.28: Arc-length parameterization of Bézier spline with parameter l

Here is the derivation of the parametrization $(\phi(t), \psi(t), \theta(t))$:

$$\begin{aligned}
\phi'(t) &= A_x T_1'(t) + B_x T_2'(t) + C_x T_3'(t) + D_x T_4'(t) \\
\psi'(t) &= A_y T_1'(t) + B_y T_2'(t) + C_y T_3'(t) + D_y T_4'(t) \\
\theta'(t) &= A_z T_1'(t) + B_z T_2'(t) + C_z T_3'(t) + D_z T_4'(t)
\end{aligned}$$

Derivation of cubic Bernstein polynomials:

$$T_1'(t) = -3t^2 + 6t - 3 \quad T_2'(t) = 9t^2 - 12t + 3 \quad T_3'(t) = -9t^2 + 6t \quad T_4'(t) = 3t^2$$

For a given arc-length parameter l (l is in fact a distance from point A on the Bézier arc) we need to find a parameter ϵ which corresponds with parameter t of the original parameterization 4.2.

$$l = \int_0^{\epsilon} \sqrt{(\phi'(t))^2 + (\psi'(t))^2 + (\theta'(t))^2} dt \tag{4.5}$$

For this integral (4.5) a primitive function doesn't exist because it contains a general quadruple polynom: $F = \int \sqrt{(at^4 + bt^3 + ct^2 + dt + e)} dt$. Thus it must be solved numerically. Our arc-length parametrization iteratively searches for ϵ for a given l. After this we put $t = \epsilon$ and we use the original parameterization 4.3 to get Cartesian coordinates of the point on the arc at given length l and corresponding parameter t.

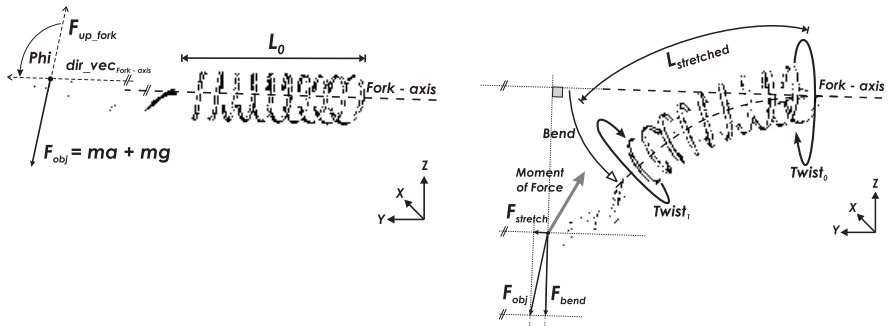
Further, we had to incorporate a compensation for changes in length of the arc (due to bending), to keep its length constant without stretching. When the object was bent without changing its length, we built the stretching ability in the deformation function. Torsion simulation is also rather complex to perform in real time, so we also made a very significant simplification for this.

It is obvious that the deformation function needs to be calibrated to a certain range of forces, resp. to the virtual world and all the objects and their weights and forces which they could produce.

We have calibrated the spring-fork tool in our mini-world for the range of object mass (0.0 vkg, 300.0 vkg), vkg = virtual kilograms. We must mention the problem of different measurement units in a virtual world and the real world. The real acceleration differs from the virtual acceleration, and certainly the force applied by the user has different scaling than the virtual one of the object. The calibration procedure takes care of this problem.

Spring-fork Parameters

There are two major inputs to the spring-fork tool. Using the tracked stylus, the user controls the initial position of the tool: the fork axis and $twist_0$. The attached object applies a force and a moment to the fork-end and causes a deformation of the spring part of the tool. See Figures 4.29(a) and 4.29(b).



(a) Initial state of the spring-fork (b) Bent, stretched and twisted spring-fork

Figure 4.29: Force-based deformation model of the spring fork

The force applied to the object F_{obj} is decomposed into the stretching force (parallel with the fork axis) and the bending force (perpendicular to the fork axis). Absolute values are calculated by using angle Phi, see Figures 4.29(a) and 4.29(b).

We designed the spring-fork tool in such a way that it is possible to choose which of the three deformation DOF will be active: bending, stretching, torsion, or any combination of these. Of course, the user can change the stiffness of each

DOF, for example to set up a stretch and a torsion resisting spring-fork which will easily bend only.

Spring-fork Manipulation

The object selection is done by inserting the spring-fork into an object. When the fork is inside, the object is highlighted. The user can then still adjust the position of the fork inside the object. To begin the manipulation a button on the stylus must be pressed. Manipulation stops when the button is released.

The manipulation procedure can be decomposed into several stages, see Figure 4.30. The initial stage is the selection. The position and orientation of the inserted fork with respect to the object will be kept during the whole manipulation. Only the spring-part of the tool will change.

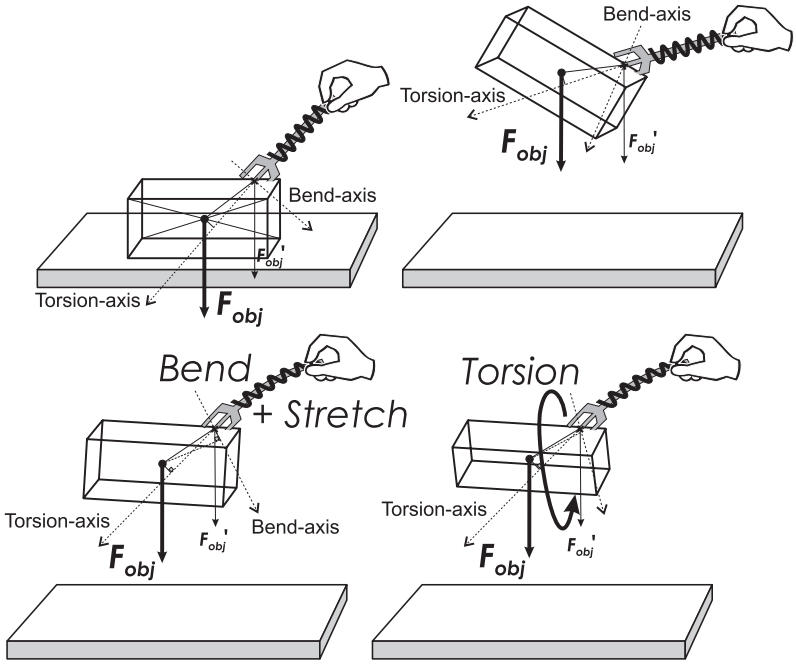
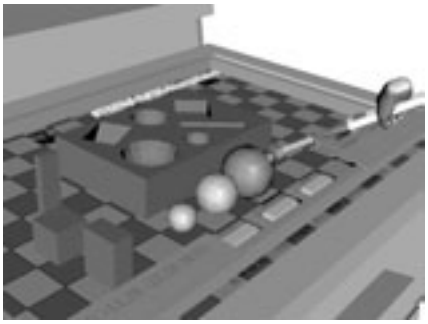


Figure 4.30: Manipulation procedure with the spring-fork

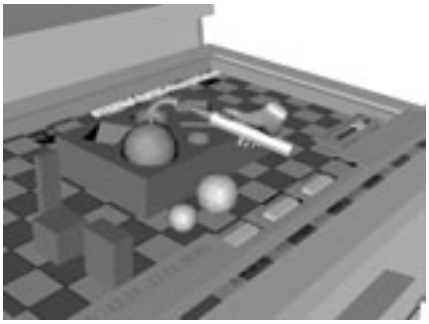
In the next stage the new position and the rotation of the user's hand are used for updating of the spring-fork. The bend and stretch deformations are performed first. After that, the torsion is calculated. This configuration (position and orientation of the object together with the deformed spring-fork) becomes a candidate for the new step, but first possible collisions with other objects have to be detected. If there is no collision, the candidate configuration will be used

to update all three spring-damper systems, and they transform the current configuration to the new configuration. The spring-dampers of the spring-fork produce at each new time step a new configuration of the object and the fork, but in the mean time the user has repositioned his hand. This triggers the next update of the system.

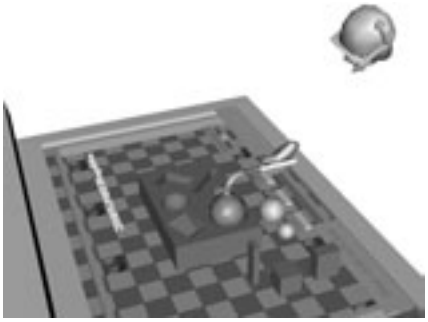
All the above steps make this simulation rather complex, especially the handling of the collisions of objects. We have used a simple method for collision checking, see Section 4.1.3.



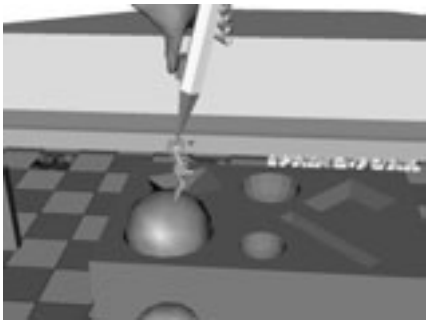
(a) Object selection with the spring-fork



(b) Object manipulation with the spring-fork



(c) Object assembly using the spring-fork



(d) Spring-fork in detail

Figure 4.31: Spring-fork demonstration (see also the Color Section)

Manipulation Experiments

To test this interaction concept, we have implemented an experimental application where the user can perform dynamic spring manipulation with virtual objects in a mini-world. The mini-world consists of several boxes and spheres which has to be assembled into the assembly box (Figure 4.31). This task can be done either with spring-based tools or with common direct manipulation tools.

The basic concept of the RWB is that the user is standing in the real world, where the physical laws apply, and is partly immersed in the virtual mini-world which is displayed on the Responsive Workbench. The user can compare the advantages of the spring-tools to the common tools (directly with the stylus or ray-casting manipulation). Finally, the user can also observe a visual feedback of dynamic behaviour of manipulated objects.

The spring-fork also provides a very clear and intuitive visual feedback of object collisions during manipulation. This flexible tool can deform according to the situation, see Figure 4.32.

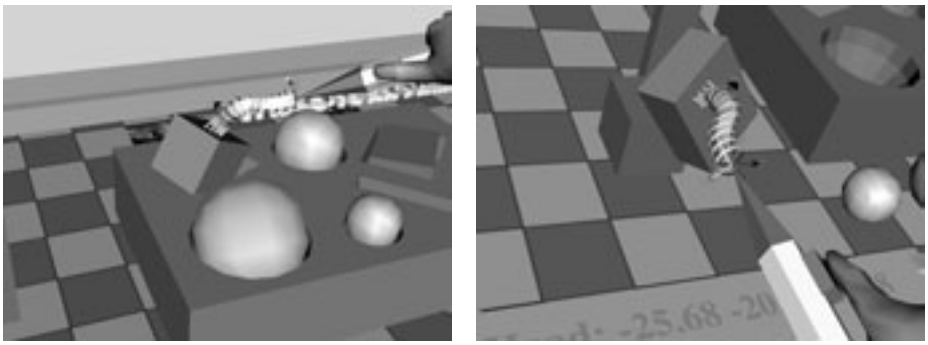
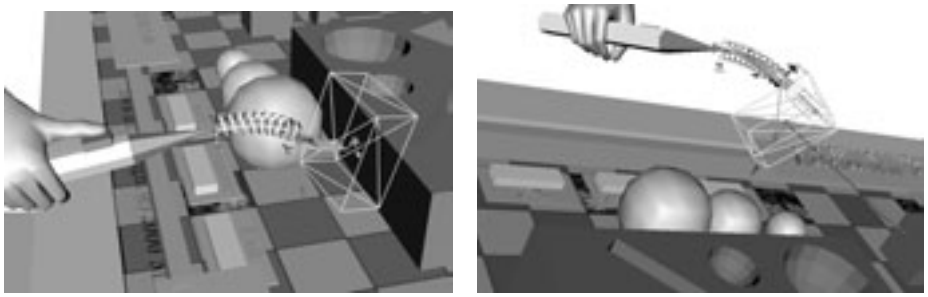


Figure 4.32: Spring-fork provides visual feedback of object collisions. The fork tool shows an S-shape deformation.

An object is selected by inserting the spring-fork into it, as shown in Figure 4.31(a). When the user has adjusted the fork inside the object, the manipulation can start. As mentioned before, a user can switch the deformation DOFs of the spring-fork on and off. For example, a manipulation with a torsion-resistant spring-fork is shown in Figure 4.33(a).



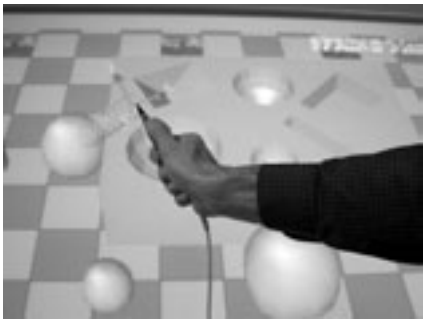
(a) (b)

Figure 4.33: (a) Torsion-resistant spring-fork; (b) Spring-fork with torsion

When all the deformation DOFs of the fork are enabled the spring-fork manipulation with the box would be as shown in Figure 4.33(b).

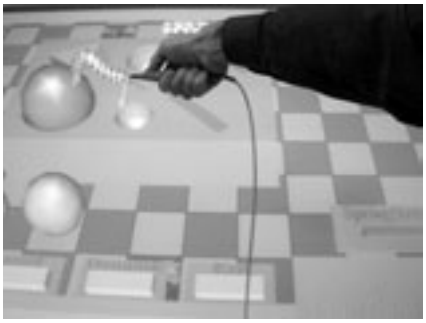
Torsion in the spring-fork and the rotation of the object are caused by the moment which is applied at the rigid fork-end. The user can compensate the rotation of the object by turning the spring-fork around its axis in the opposite direction. The spring-damper of the torsion parameter will take care of the transformation of the rotation to the attached object.

(a) RWB overview: Spring tools used for the assembly task



(b) RWB detail: Spring manipulation

(c) RWB detail: Spring-fork manipulation



(d) RWB detail: Spring-fork manipulation

Figure 4.34: RWB-view: Spring-tools demonstration on the Workbench

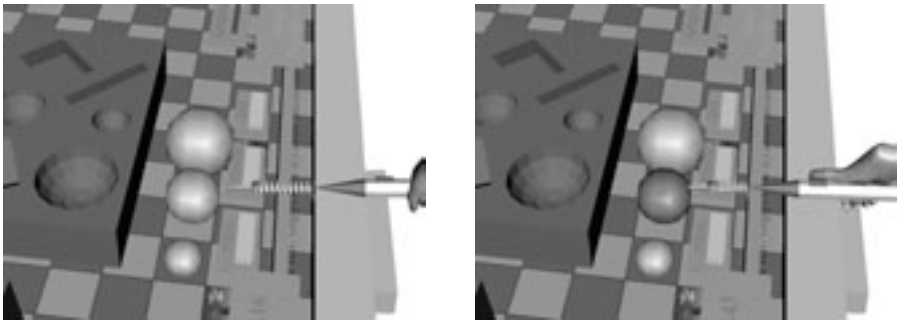
Figure 4.34 shows snapshots of the Responsive Workbench with a user performing the object assembly task using the spring tool and the spring-fork, respectively.

4.2.5 Other Spring-Based Tools

As described before, the deformation DOF's of the spring-fork can be disabled. This way we have derived other spring-based tools.

Spring-probe

By disabling the torsion and bending DOF's and use of a modified tool tip we have created a spring probing tool. The spring-probe can be used for probing the material stiffness of an object, local deformation of an object, or simply pushing an object. This tool has only 1 DOF (length) and can control 3 DOF (xyz) or 1 DOF (i.e. pressure) of an object, see Figure 4.35.



(a) Object selection with the spring-probe (b) The spring-probe is pushing to the object

Figure 4.35: Spring-probe demonstration

Although this tool seemed also quite promising we did not experiment much with it. Instead, we have invested our research time in the next tool, which is an instrument for particle steering in Molecular Dynamics simulations.

Spring Particle Manipulator

One of the drawbacks of the spring manipulation tools, described above, is that they work only within direct reach of the user's hands. To overcome this we looked into the real world and we have made use of the fishing rod metaphor. In a fishing rod the principles of the spring-damper are also hidden. Usually, the fishing tool can bend only. We have adapted this metaphor and created the Spring Manipulator, which can stretch and bend. Instead of using a nylon line we actually insert the tool itself into a manipulated object.

The object is selectable from any distance using the ray-casting and then manipulated with this Spring Manipulator. We have applied this technique in our MolDRIVE system, where we can perform particle steering and exert an external force on particles in this way (Figure 4.36). More about MolDRIVE and particle steering can be found in Section 4.3 and 6.2.

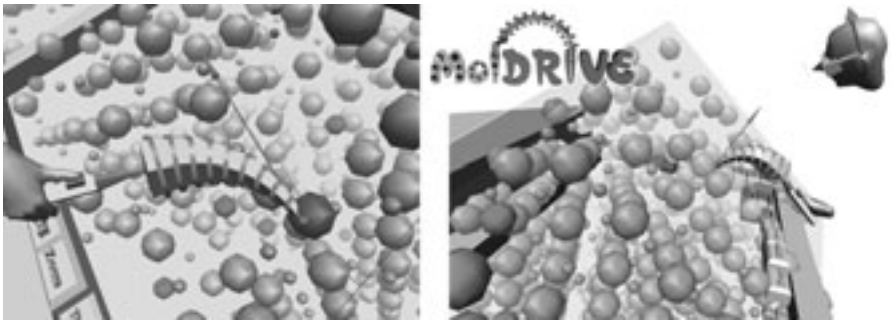


Figure 4.36: Spring Manipulator for particle steering in MolDRIVE

4.2.6 Visual Force Feedback: Summary and Discussion

We have developed a scheme of dynamic object behaviour for manipulation of objects in virtual environments. We have presented and described in detail the set of spring-based manipulation tools, with which we can control both translational and rotational motions. These tools were designed to produce a realistic visual force feedback.

We performed informal tests of object manipulation with a group of VR experienced and non-experienced people. The spring tools were easy and intuitive to use and also the subjects performed better with the assembly task. The results show that object behaviour appears more natural and predictable than the 'unphysical' objects in most virtual environments.

However synthetic the models of the spring-tools are, they look and feel surprisingly real. The approximation of the mechanics seems to be good enough to create the illusion of mass and substance. Also the performance of the dynamic simulation keeps up with the tracker speed (50Hz). Of course, the actual performance and the frame rate varies with the amount of collisions and the complexity of collision behaviour, which seems to be a bottleneck.

We can easily define a long list of extensions for future work in this area. The numerical computations for dynamic simulation can become very complex, and may put serious limits on performance. Therefore, we have greatly simplified the underlying physics. The simulation of the contact behaviour, efficient collision detection and handling, and constrained motion are areas of future work and improvement.

We have successfully applied the spring tools in particle steering of Molecular Dynamics in our MolDRIVE system, see Section 4.3. Haptic devices that could do the job of particle steering in the same free and easy fashion to use as the Spring Manipulator do not exist, and may not exist in the near future.

4.3 Particle Steering Tools for Molecular Dynamics

In this section we present new virtual spring manipulator-based tools for steering particles in molecular dynamics simulations from virtual environments. We briefly review the MolDRIVE system, our visualization and computational steering environment for real-time molecular dynamics simulations, which is the platform for our particle steering implementation.

Our study concentrates on visual feedback tools. We compare a basic virtual particle steering method with two other methods using a spring manipulator. The first just creates a visual feedback of a flexible connection between the user's interaction device and the steered particle, while the second technique creates a visual illusion of force feedback. The user can, through the spring manipulator, exert a force on the manipulated particle in the MD simulation.

4.3.1 Introduction

Current trends in molecular dynamics (MD) visualization show an increasing importance of interactive steering capabilities for MD simulation systems. Due to the increasing computational power we can simulate larger MD problems in real time. But there will always be enough MD simulations where we would have to wait days or weeks for results. Thus many of today's MD systems [Web-MD-demmpsi; Web-MD-gromacs; Web-MD-namd] support visualization and steering of running simulations.

For many reasons, atomic or particle steering seems to be a very attractive function of MD simulation and visualization systems. It can be used for protein design [Arthur *et al.*, 1998] and for molecular docking. With the particle steering the user can make a particle overcome energy barriers within the simulated system. This way a desired configuration can be reached in a shorter time, which is very useful for studying specific energy system configurations. Based on previous research of the Computational Physics group at TU Delft, we perform particle steering on real-time simulations of a β-Alumina electrolyte system [Beckers, 1999].

VR systems offer a stereoscopic 3D immersion into the microscopic scale molecular environments. The 3D tracking technology gives us the opportunity to interact with these virtual environments. Specialized VEs were developed for Interactively Steered Molecular Dynamics (ISMD) [Prins *et al.*, 1999; Arthur *et al.*, 1998], where the researcher can control the simulation as well as interact with the particles, allowing to get in touch with this micro-environment. However, the existing implementations of particle steering techniques for VEs are mostly based on using the haptic devices, often constraining user interaction. With our virtual particle manipulators we want to show a very good alternative to the existing approaches.

Our proposed particle steering methods offer visual feedback, showing if the performed interaction with the particle is physically valid. Each user interaction with a particle is validated by the simulation program.

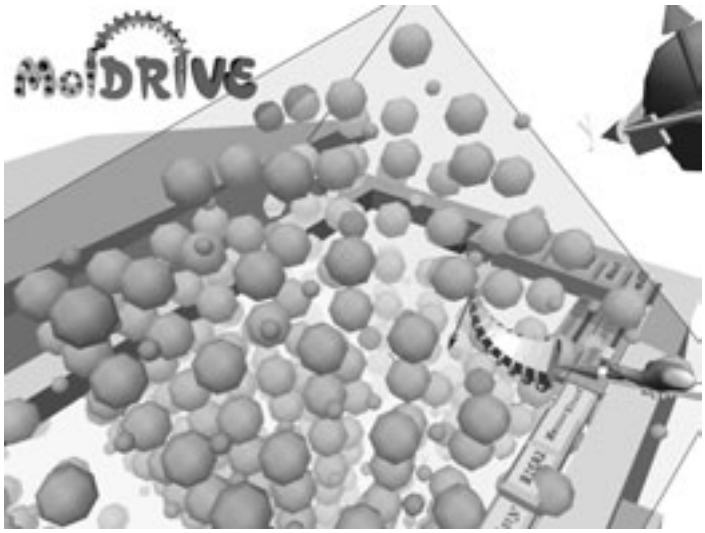


Figure 4.37: Visual force feedback using spring manipulator

Currently, haptic devices can be used to obtain an input position and force from the user, as well as to render the force feedback.

Haptics adds to VEs a dimension of touch and force feeling; the user can feel a real force during pulling or pushing objects. Haptics is also used to provide the force input and the force feedback during atomic steering [Stone *et al.*, 2001]. Most of the PHANToM - like haptic devices are used in desktop systems, and only a small number of them is used in immersive VEs [Grant *et al.*, 1998; Taylor II, 1999; Brederson *et al.*, 2000; Durbeck *et al.*, 1998]. Some MD visualization systems use the haptics to apply force on particles or to feel the forces of the simulated system. Advantage of VEs is that they produce greater 3D immersion and much more natural interaction with the MD simulation than the desktop systems.

The motivation of our research is to prove our concept that visual force feedback is sufficient for effective particle steering in MD simulations. We present a graphical force-feedback method and we provide a visual interface as an emulation of direct force input. We have experimented in the MolDRIVE system with the following three particle steering methods:

- **Virtual particle** displays a virtual particle on a user-requested position and the original particle on a position accepted by the simulation. The new particle position is sent to the simulation and checked for validity.
  More about this technique in Section 4.3.3.
- **Spring feedback** displays a bending spring between the stylus and the manipulated particle on an accepted position. The virtual particle with the ray shooting from the stylus show the desired position. The spring

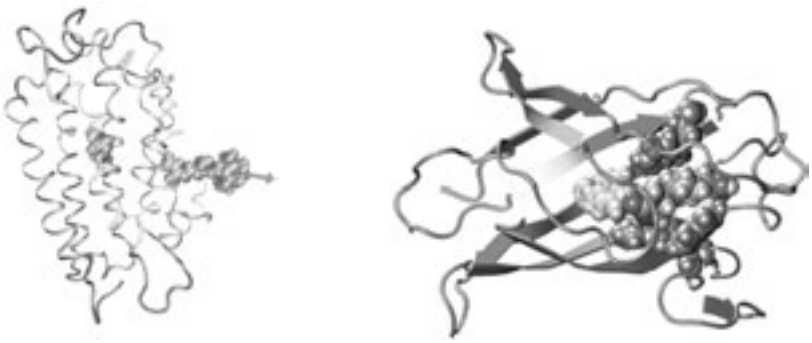
manipulator forms only a flexible visual connection between the stylus and the manipulated particle. No force is calculated from the deformation of the spring in this case. More about this technique in Section 4.3.4.

- **Visual spring force feedback** displays a bending spring between the stylus and the particle; the 2 DOF (degrees of freedom) spring deformation (stretching and bending) defines an external force acting on the particle, see Figure 4.37; the force is sent back to the simulation. More about this technique in Section 4.3.5.

We first review some related work on MD simulation in VE, on particle steering, on force feedback and on the usage of springs as a visual feedback instrument. This work follows the initial implementation of the spring manipulation tools [Koutek & Post, 2001c; Koutek & Post, 2000]. Then we provide an overview of the MolDRIVE system, and we describe in detail the three particle steering methods. This section documents their development and experiments in various MD applications.

Related Work

Molecular Dynamics simulations are used to study the properties and behaviour of complex particle systems by solving Newton's equation of motion numerically for all particles for a relatively small time interval (10^{-15} sec). The growing interest for being able to steer an MD simulation has resulted in the development of several systems such as Steered Molecular Dynamics (SMD) [Leech *et al.*, 1996] and Interactive Molecular Dynamics (IMD) [Stone *et al.*, 2001].



(a) (b)

Figure 4.38: (a) Steered MD; (b) Interactive Molecular Dynamics (IMD);

Initially, we have developed the MolDRIVE system on top of the DEMMPSI simulation program [Web-MD-demmpsi]. Because of its good computational speed compared to other MD programs [Beckers, 1999], and its parallel implementation using MPI, it is a good candidate to perform atomic steering in real-time Molecular Dynamics simulations.

Usually haptic devices [Grant *et al.*, 1998; Taylor II, 1999; Brederson *et al.*, 2000; Durbeck *et al.*, 1998] are used to add user force on particles and to experience the reaction of the simulation. In SMD external forces are implemented with springs between the target position and the restrained atom. Visual Molecular Dynamics (VMD) [Humphrey *et al.*, 1996] is commonly used to perform the 2D or 3D visualization. VR extensions to VMD have been presented [Prins *et al.*, 1999; Arthur *et al.*, 1998], where the user's immersion into the VE is supported by head-tracking with ability to interact with the visualization.

MolDRIVE uses the Workbench as a Virtual Reality visualization environment with spatial interaction. It uses a metaphor of a laboratory table, where the user performs an interactive experiment with MD system. What is specific in our case, is the use of the DEMMPSI simulation, and our new contribution is the particle steering tools, which use the spring manipulator.

The springs are very well established phenomena in the world of physics and graphics as well. Springs were used by other researchers for manipulation of objects on the Responsive Workbench [Fröhlich *et al.*, 2000]. The spring particle manipulators are based on our previous work, the spring manipulation tools [Koutek & Post, 2001c; Koutek & Post, 2000], where we have demonstrated the use of spring for manipulation of objects in virtual environments in a simple assembly task. The spring-fork [Koutek & Post, 2001c] needs just one hand with the stylus to control position and rotation of attached object, and it provides a clear visual force feedback.

4.3.2 Molecular Dynamics Real-time Virtual Environment

We have developed a system named **MolDRIVE** , which represents a virtual environment for visualization and steering of real-time molecular dynamics simulations. MolDRIVE is in fact a visualization and computational steering environment with an interface to several MD simulation programs, which run in parallel on remote supercomputers.

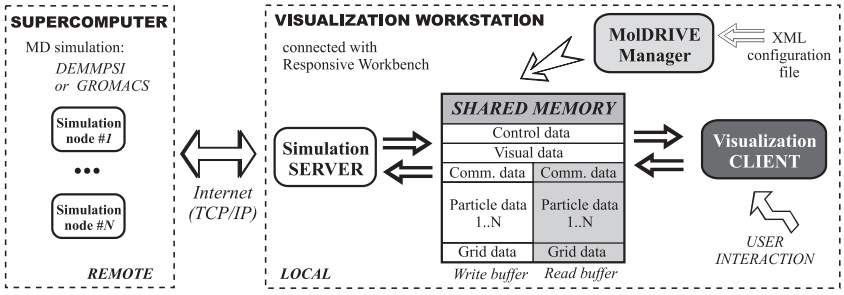


Figure 4.39: MolDRIVE overview: remote parallel MD simulation is running on a (parallel) supercomputer. Visualization and computational steering is done on the RWB which is driven by a powerful graphics workstation (SGI Onyx2).

Currently, it uses DEMMPSI [Beckers, 1999; Web-MD-demmpsi] and also Gromacs [Web-MD-gromacs]. Remote simulations allow the high performance graphics workstation, to fully concentrate on the visualization part of the process and larger systems can be visualized. The MolDRIVE system layout is shown in Figure 4.39. The MD simulation, in this case DEMMPSI, runs remotely on parallel supercomputers such as a CRAY T3E (128 proc.), an SGI 1100/1200 Beowulf Linux cluster (54 proc.) or an SGI Origin 2000 (8 proc.). Gromacs is not running in parallel using MPI yet; the version used supports only PVM.

MolDRIVE System Components

The simulation server is a thread-based program, which communicates simultaneously through TCP/IP sockets with each MD simulation node on the supercomputer. The simulation server runs on an SGI Onyx2. At each time step the server transmits a data request of the visualization client to each simulation node. Each node handles this request and sends the simulation data back to the server. This data is then placed in shared memory. The system architecture allows transmission of user interactions back to the simulation.

The visualization client also runs on the SGI Onyx2. The RWB visualization client is implemented using the RWB-Library [Koutek & Post, 2002; Web-RWB-lib] and is based on Iris Performer and OpenGL. It runs in parallel on all 4 processors of the Onyx2 to maximize the speed of the visualization pipeline and to keep up with the interactive visualization frame rates.

The visualization client creates a graphical representation of the simulated system and updates the particle position, forces and velocity vectors. It can also visualize the derived grid data from the simulation, such as particle densities, potentials, kinetic energy, etc. Through the 3D GUI the user has access to several visualization and steering tools.

Shared memory is used to communicate and exchange simulation and communication data between the visualization client and the simulation server.

The MolDRIVE manager reads an XML configuration file with the description of the simulation and the visualization environment. It initializes the shared memory data structures. It also activates and controls the RWB visualization client and the simulation clients (through the simulation server).

The shared memory consists of *particle data*, such as position, velocity and forces, *grid data* such as particle density or potential, and the *communication data*, which are used to steer the simulation and to get feedback data. We use double buffering to speed up the data throughput. The *control data* is used to manage the read and write access from both visualization client and simulation server to the shared memory. RWB visualization client reads the data from the *read buffer*, while the simulation server fills in the *write buffer* with simulation data for the next time step. When both have finished the MolDRIVE manager switches the pointers and tells to the visualization client that new data is ready.

Another benefit of this method is that it reduces the effects of network latency. A small disadvantage is that during particle steering the write buffer is filled with simulation data which are one step behind.

For the MolDRIVE system, we use a Na^+-β-alumina simulation as an example, which runs in real time at reasonable frame-rates, up to 24 Hz, depending on the number of particles in the simulated system and the complexity of inner system relations and interactions. This Na^+-β-alumina electrolyte is studied because of its fast ionic conduction properties. We usually run this simulation on up to 8 nodes in parallel on the supercomputer. The simulation speed scales up quite acceptably with the number of processors.

DEMMPSI uses domain decomposition of problem space; thus, each processor has to simulate only the particles which fall within its sub-domain. In case of long range interaction forces with particles from other sub domains communication with other simulation nodes takes place. For systems containing 2000 up to 4500 particles, we have found experimentally that 8 nodes seems to be an optimal number for DEMMPSI. If more processors are used, communication between the computational nodes slows down the whole simulation. From our experience, if the refresh rate does not drop much below 8 Hz, we can still interactively perform the particle steering even with a system containing more than 2000 particles.

On top of the MolDRIVE system we have implemented three particle steering methods, which we will describe in the following three sections.

4.3.3 Virtual Particle Steering Method

The virtual particle method and its enhanced version with the spring feedback (Section 4.3.4) send a newly requested particle position through the control data buffer via the simulation server to the simulation nodes. The simulation then evaluates the new position in a new simulation time step. If this new position doesn't destabilize the simulated system, the new data are sent to the simulation server and then placed into the shared memory where the visualization client reads its data.

Underlying Physics

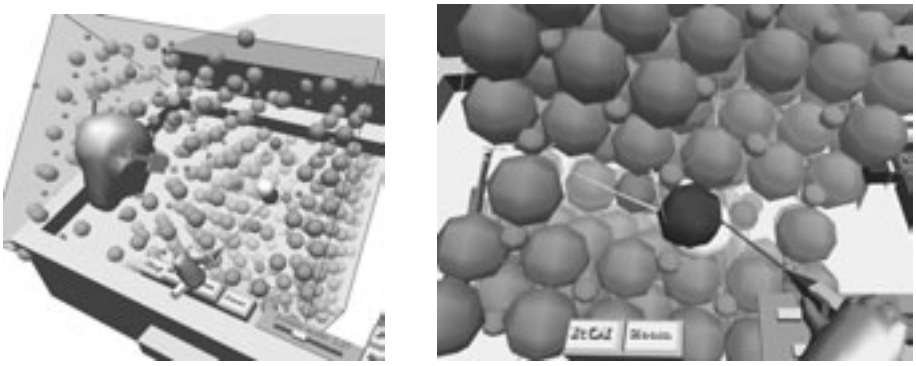
During manipulation of a particle through the system, the potential energy of this particle changes and may reach a value which the particle normally could not get. Such a situation occurs for example when the user tries to place a particle very close to another particle. This could cause unpredictable behaviour and eventually resulting in a simulation crash. To prevent this, the Boltzmann constant from formula 4.6 is used. It gives us a relative probability P, that the system will move to a configuration with an energy value at a distance ΔE from the current energy.

$$P = exp(\frac{-\Delta E}{k_B T}), \qquad \Delta E \geq 0 \tag{4.6}$$

Where k_B is the Boltzmann constant and T is the temperature of the system. For the reference energy the particle energy at the first position of the interactive movement is used. A new particle position is considered to be valid if the corresponding energy at that position could occur with a probability that is larger than or equal to a probability threshold α, which is usually chosen to be in the order of 1%.

Practical Usage

The virtual particle steering method is shown in Figure 4.40. The user selects a particle using ray-intersection. At the moment of the first click on the stylus button, the transformation from the stylus to the particle is computed. Next time, when the RWB-library updates the position and orientation of the stylus, the new position of the particle will be computed using this transformation. At this position, at the end of the ray, a virtual (blue) particle is displayed. It indicates a new desired position of the particle. This position is sent to the MD simulation for validation. The white particle shows this particle on the last valid position that was accepted by the simulation program.



(a) (b)

Figure 4.40: Virtual particle method: (a) in overview; (b) in detail

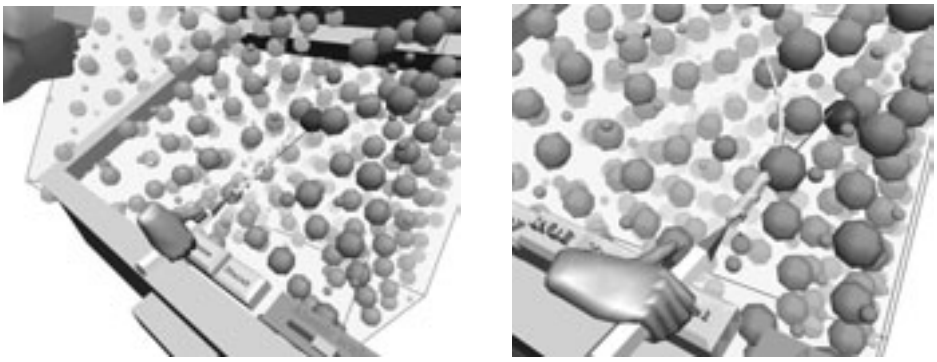
Figure 4.40(b) shows the steering of the Na^+ atom through the conduction layer of the electrolyte. The velocity of particles as well as the forces acting on them are visualized using arrows. They react to the user's interaction giving an immediate visual response.

The virtual particle always follows the motions of the stylus, while the accepted white particle only moves when the simulation allows this. Sometimes it results in a very inconsistent view. For a user manipulating a particle, it seems as though the original particle was left behind at its place and some new particle pops up and follows the ray. With increasing distance from the white particle also the visual gap between them increases.

This is the situation where the spring manipulator can be used. It will connect the stylus with the manipulated particle at its last valid position.

4.3.4 Spring Feedback Particle Steering Method

This method visually differs from the previous one in the way the user experiences the acceptance of the new position by the simulation. During particle manipulation, the user's stylus is always connected to the selected particle through a spring manipulator, as shown in Figure 4.41. The position of the manipulated particle is equivalent to the last position accepted by the MD simulation. The acceptance is calculated in the same way as in the previous method.



(a) (b)

Figure 4.41: Spring feedback method: (a) in overview; (b) in detail

This spring based interaction tool has 2 DOF for deformation: stretching and bending. Next it has 6 DOF in position XYZ and orientation HPR (Head, Pitch and Roll), which is given by the tracking of the stylus. The spring manipulator is attached to the tip of the stylus and points always in the Z-direction of the stylus's local coordinate system. The other end of the spring manipulator is always attached to the manipulated particle at the point of the initial ray intersection.

At the moment of the first button click, a transformation from the center of the particle to the point of initial intersection is calculated, see Figure 4.43. Next time, when a newly accepted particle position arrives at the visualization server and the RWB-library reads the new stylus position and orientation, the new intersection point is calculated by transforming the actual particle central

position with the pre-computed transformation. The spring manipulator will bend and stretch to end precisely at the new intersection point, see Figure 4.43.

This scheme is similar to the original spring-fork [Koutek & Post, 2001c] with 3 deformation DOF (stretching, bending, torsion). The spring-fork can control position and orientation of objects with 6 DOF, see Figure 4.29 in Section 4.2.4.

The new spring manipulator has only 2 deformation DOFs. It bends without torsion and controls only the position of an object. This tool has only a single point attachment with the particle, as there is no need to apply rotation on single particles during steering. The effect of the visual spring is that the user never loses a particle during manipulation. It stays visually connected to the stylus, no matter how the user deforms the spring, see Figures 4.41 and 4.42. The new (desired) position which is sent to the simulation is indicated by a virtual particle at the end of the stylus ray just as in the previous method. The spring metaphor makes the user feel that the particle is being dragged to a new position, which is experienced as more natural than the previous steering method.

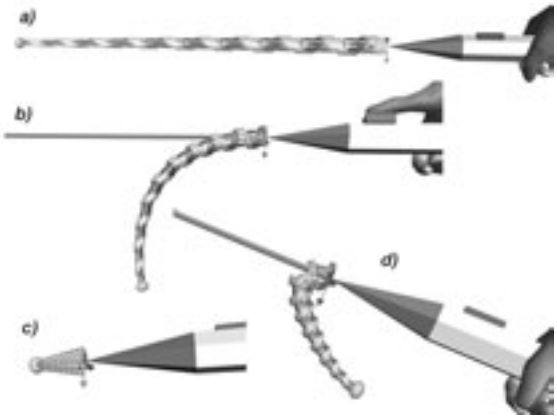


Figure 4.42: Spring manipulator - experimental deformations: a) pulling / extension b) bending c) pushing / compression d) extreme pushing

4.3.5 Spring Force Feedback Particle Steering Method

The previous method provides only a visual contact with the manipulated particle. The feedback is based only on particle positions. Sometimes, it results in irregular motion of the particle, as the new particle positions may be accepted or rejected by the MD simulation.

A way to avoid this is to use the force input and the force feedback delivered by the spring instead. The spring manipulator provides a visual force feedback as well as an emulation of direct force input through the idea of bending and stretching a spring.

Model of the Spring Manipulator

The graphical model of the spring manipulator consists of a spring, a beam column inside and a small sphere at the tip of the instrument, which supports the idea of a single point contact with the manipulated particle, in contrast to the original spring-fork, which has contact with intersected objects at four points, and therefore can also control rotation of objects. This was not needed for a single particle steering. Thus the spring manipulator became more simple. As mentioned before, the spring has 2 deformation DOFs: stretching and bending. Figure 4.42 shows in wireframe: a) stretched spring b) regular bending c) highly compressed spring d) extreme spring deformation.

The geometric model of the spring manipulator is kept simple, with a small number of vertices, because the vertices and normals of the facets have to be transformed according to the spring deformation. This must be done each time when the RWB-library reads a new position and orientation of the stylus (48Hz).

Spring deformation follows the arc-length parameterized cubic Bézier curve segment which is defined by the 4 control points: A,B,C,D, see Figure 4.43.

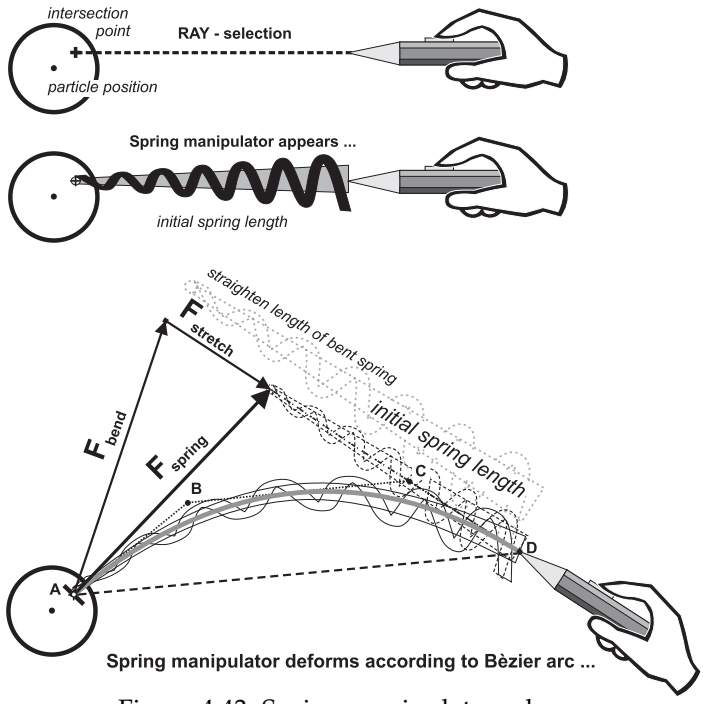


Figure 4.43: Spring-manipulator scheme

More about the deformation model of the spring tools (spring and spring-fork) can be found in Section 4.2.4.

Stretching and bending of the spring manipulator causes a reaction of exerting a force to restore its initial length and straight form. The total spring force consists of the stretching and the bending component:

$$\begin{aligned} \text{Spring stretching force:} \quad & \overrightarrow{F}_{stretch} = -k\,\overrightarrow{\delta}_{stretch} \\ \text{Spring bending force:} \quad & \overrightarrow{F}_{bend} = -k\,\overrightarrow{\delta}_{bend} \\ \text{Total spring force:} \quad & \overrightarrow{F}_{spring} = \overrightarrow{F}_{stretch} + \overrightarrow{F}_{bend} \end{aligned} \tag{4.7}$$

Where: k= spring constant, $\overrightarrow{\delta}_{stretch}$= relative stretching of the spring, $\overrightarrow{\delta}_{bend}$= relative bending of the spring

Calibration of the Spring Manipulator

The spring manipulator is calibrated by the spring constant and by setting the initial length of the spring. In the rest position it has a straight form. The spring manipulator has two ends: one is permanently connected to the stylus and the other end is connected with the particle at the point of the ray intersection (see Figures 4.43 and 4.44).

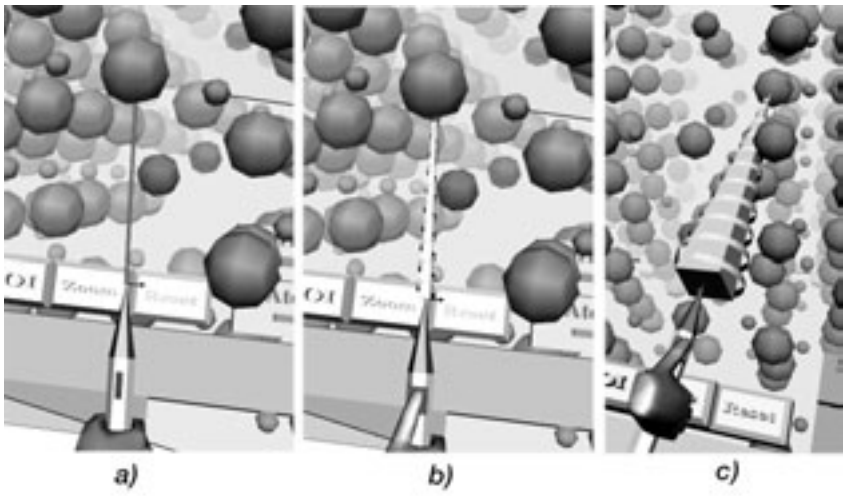


Figure 4.44: (a) Ray selection of particles and initial calibration of the spring manipulator (b) spring constant $k = 1eV/\text{Å}^2$ (c) $k = 5eV/\text{Å}^2$

A moving particle in a running MD simulation can be selected with the ray casting technique, see Figure 4.44(a). When the user clicks the button the initial spring length is computed, the ray disappears and the spring manipulator is displayed instead. The spring strength and dimensions are based on a given spring constant. Figure 4.44(b) shows $k = 1eV/\text{Å}^2$ and Figure 4.44(c) shows $k = 5eV/\text{Å}^2$ (common units in MD community). The user can change the spring constant with the slider, see Figures 4.45 and 4.46.

Particle Force Model of the MD Simulation

The original force model of the particle of DEMMPSI had to be adapted in order to enable the external user force. The force that the user exerts on a particle is also sent to the simulation and is added to the force delivered by the other particles due to long and short range interactions:

$$\overrightarrow{F}_{part} = \overrightarrow{F}_{short} + \overrightarrow{F}_{long} + \overrightarrow{F}_{user} \tag{4.8}$$

In the spring force feedback method we set $\overrightarrow{F}_{user} = \overrightarrow{F}_{spring}$. After the force calculation the particle positions are updated. Without taking precautions, the influence of the extra force to the simulation will finally result in a temperature change and a possible drift of the simulation which is caused because of a non-zero total momentum of the system. Therefore, after finishing the particle manipulation, the velocities of the particles are re-scaled in order to restore the original temperature and to maintain zero momentum.

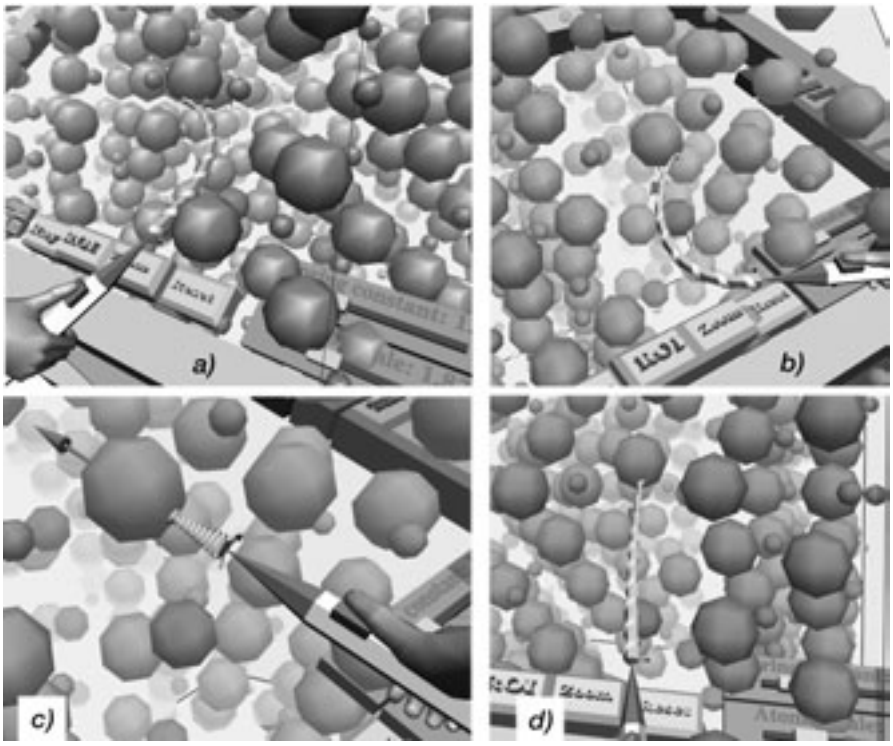


Figure 4.45: Real-time particle steering using force feedback of the spring manipulator with spring constant $k = 1eV/\text{Å}^2$

Examples of Visual Force Feedback Particle Steering

In Figures 4.45 and 4.46, the spring manipulator is used to steer the particle in a running MD simulation. After particle selection and initial auto-calibration of the spring manipulator, the user can exert an external force on the particle through the visual spring mechanism into the force model of the MD simulation.

The arrow at the end of the spring shows the spring force which is computed from the spring deformations. Figures 4.45a and 4.45b show a user steering the Na^+ ion to the right or to the left respectively. The user is trying to find a new equilibrium position for the selected particle. Figures 4.45c and 4.45d demonstrate push and pull techniques.

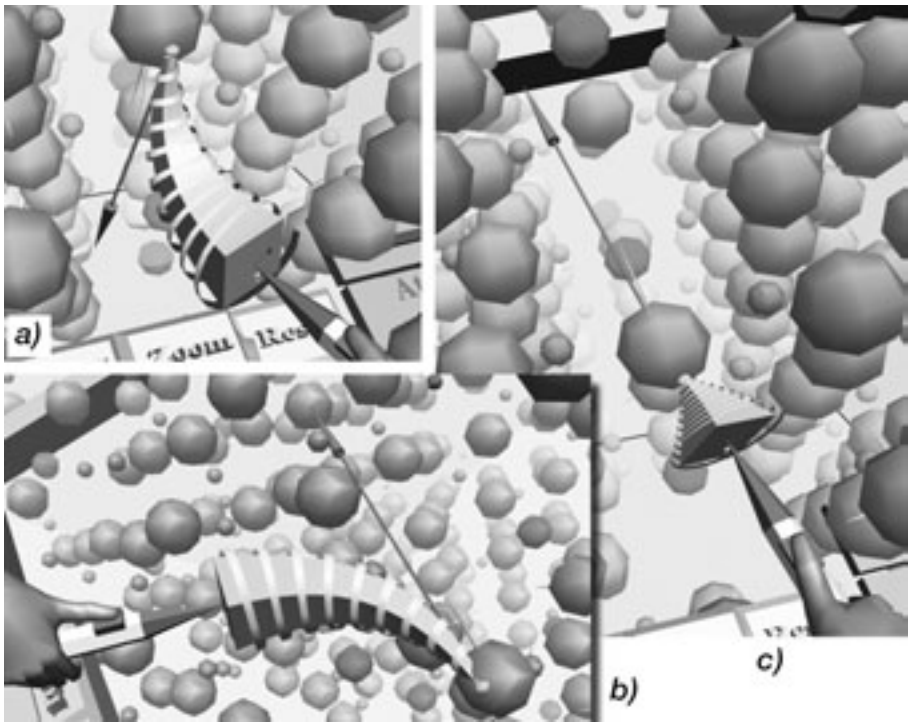


Figure 4.46: Real-time particle steering using force feedback of a strong spring manipulator with spring constant $k = 5eV/\text{Å}^2$ (see also the Color Section)

In Figures 4.45 and 4.46, we can see a β-alumina system consisting of 2088 particles. There are atoms of sodium Na (red), oxygen O (yellow) and aluminum Al (small blue particles), see also the Color Section. The conduction plane is formed by the sodium ions. In Figure 4.46c the user looks down through this conduction plane.

In our visualization we use a depth cuing technique based on distance-haze to improve the depth perception in a 3D visualization on the Responsive Workbench. It significantly improves orientation in the stereo images on the RWB (there we use a "black fog"; darker with distance) as well as the RWB simulator images in this section (here we have used a "white fog"; lighter with distance). It helps a lot to distinguish between particles in the foreground and in the background in addition to the stereo-effect.

Figure 4.46 shows an Na atom being driven by a strong spring manipulator. When a weak spring is used, the spring force will be relatively small to steer the particle in the desired direction. With a strong spring is it possible to drag a particle even against strong atomic binding forces. In this particle visualization the user can also adjust the size of the displayed particles to see through the system. When the *Van der Waals atomic radius* is used, the particle system is so dense that the user cannot look through it. Our MD simulation program provides the visualization client also with particle forces (green arrows) and velocities (white arrows), which are also visualized, see also the Color Section.

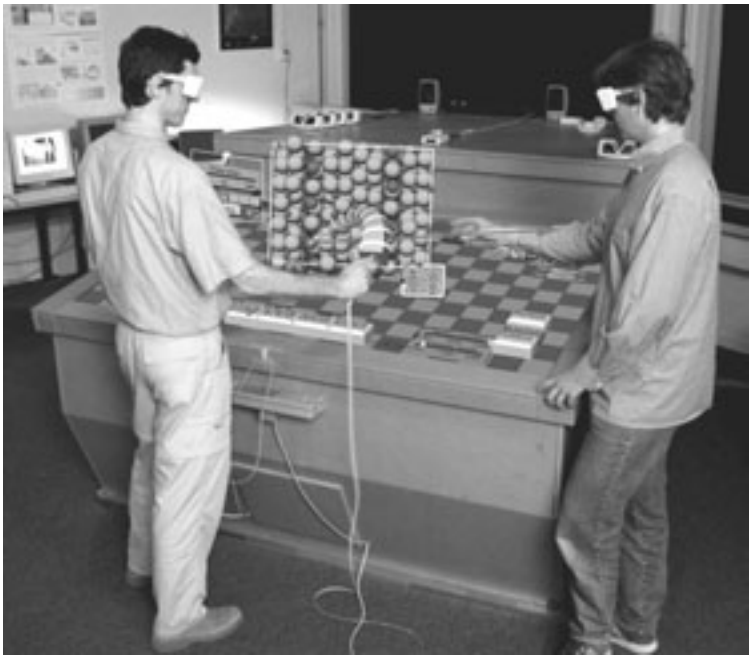
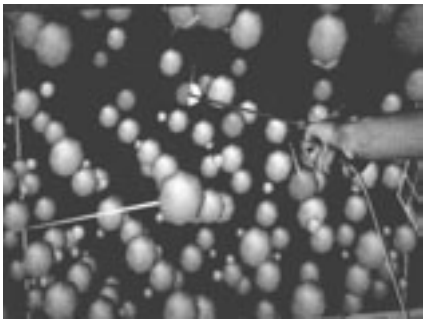


Figure 4.47: Particle steering of β-Alumina electrolyte with the spring manipulator on the Workbench (see also the Color Section)

A user performing particle steering in the MolDRIVE environment gets a complex visual feedback of the interactions with the particle, see Figures 4.47 and 4.48.

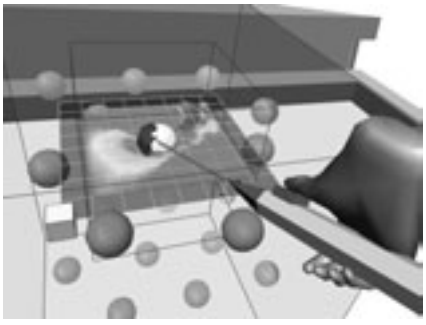


(a) User performing steering with virtual particle method

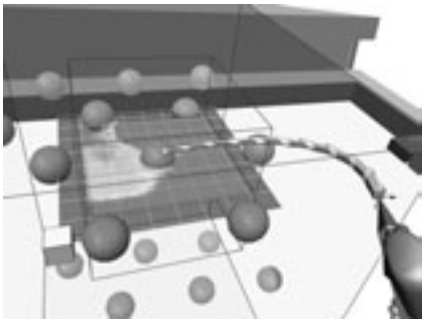
(b) Particle steering using the spring manipulator;

Figure 4.48: RWB photo: MolDRIVE - particle steering (DEMMPSI); the user moves the sodium atom through the conduction plane, the other non-sodium atoms are not displayed (b). The large arrow at the end of the spring manipulator shows the spring force and the small arrow shows the resulting velocity of this particle.

Optionally, we can interactively visualize derived grid data of the whole system, such as kinetic energy, particle densities or particle potential, which shows important properties for particle steering (see Figure 4.49).



(a) Virtual particle steering method



(b) Spring force feedback steering method

Figure 4.49: MolDRIVE (DEMMPSI): particle steering; A color data-slicer is used to display the potential around the particle.

An observation we have made is that during particle manipulation the user normally introduces some local disturbance to the simulation. We have found that relaxation of the system takes about 10 to 30 fs (femto seconds), which means about 2 seconds during VR visualization at the Workbench with the DEMMPSI simulation running at 15Hz with a timestep of 1 fs.

4.3.6 Particle Steering: Summary and Discussion

We have presented steering methods which enable users to interact with particles in real-time MD simulations on the RWB. The performance of the MolDRIVE system allows to steer in real-time systems with up to 4500 particles.

The virtual particle and spring feedback particle methods can be used to navigate a particle through relatively low energy barriers with little disturbance of the simulation. The visual spring helps users to keep contact with the steered particle and is experienced in a more natural way.

The virtual particle with the spring feedback is also a useful technique especially when the user wants to check whether the particle could be placed at a given position rather than to be dragged to that position, which is the effect of the spring force feedback method.

Comparing the three steering methods, we have found that the spring force feedback method provides the smoothest way of steering the simulation to a desired configuration. Further, it seems that adding the force delivered by the spring manipulator to the particle makes sense regarding the underlying physics. The user can visually experience the strength of inter-atomic forces and the amount of work is required to reposition the atoms.

The researchers studying the β-Alumina electrolyte can now interactively perform experiments using the Responsive Workbench with the MolDRIVE system. The observations that are made in this virtual MD laboratory, when performing these experiments, help to create a better understanding of underlying processes. They also help to evaluate the simulation methods and to move the fundamental MD research further.

We have tested the spring force feedback technique on steering of polymer simulations. A simple example of a carbon polymer is shown in Figure 4.50. Initial results indicate that the techniques can be easily extended to very different types of MD simulations.

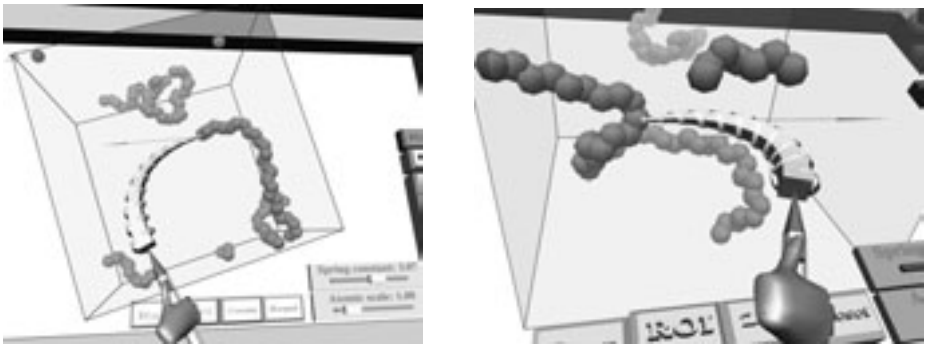


Figure 4.50: MolDRIVE (DEMMPSI): steering of a simple carbon polymer

We have also used MolDRIVE to study protein simulations using the Gromacs MD simulation program, see Figures 4.51 and 4.52.

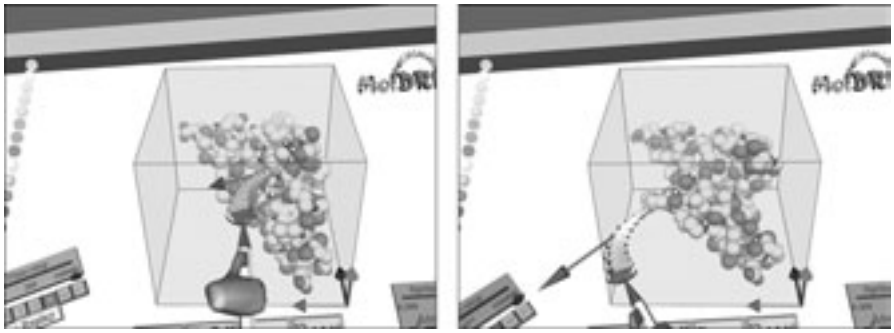


Figure 4.51: MolDRIVE (Gromacs): particle steering of T0099 protein

Steering particle (atomic) groups and exploring the possibility of extending our spring manipulators to move atomic groups were identified as items for future work.

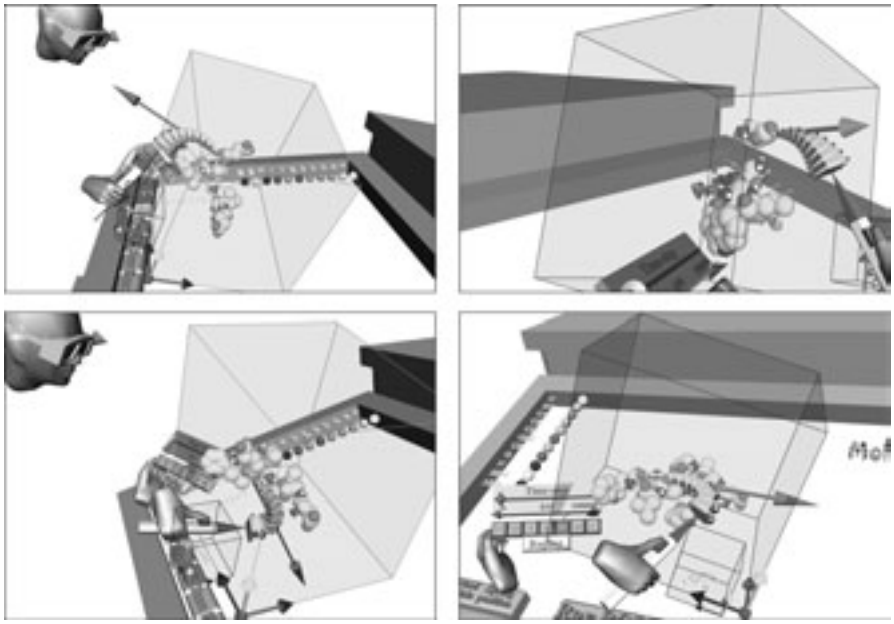


Figure 4.52: MolDRIVE (Gromacs): particle steering of protein fragment

We believe that we have also reached our initial goal to make all the particle steering tools intuitive and easy to use for non-VR experts.

Chapter 5

Exploration and Data Visualization in VR

This chapter describes our approach to interactive visualization and exploration of data in VR. Interaction with the VE plays an important role in immersive visualization. From this reason, we describe first suitable interaction scenarios and techniques, see Section 5.1. These interaction and exploration concepts were implemented in VRX, our visualization toolkit for the Virtual Workbench, which is described later in Section 5.2.

5.1 Towards Intuitive Interaction and Exploration

In this section we present a basic set of intuitive exploration tools for data visualization in a virtual environment on the Responsive Workbench. First, we introduce the Plexipad, a transparent acrylic panel which allows two-handed interaction in combination with a stylus. After a description of various interaction scenarios with these two devices, we present a basic set of interaction tools, which support the user in the process of exploring volumetric datasets. Besides the interaction tools for navigation and selection, we present interactive probing tools which are used as an input for complex visualization tools and for performing virtual measurements.

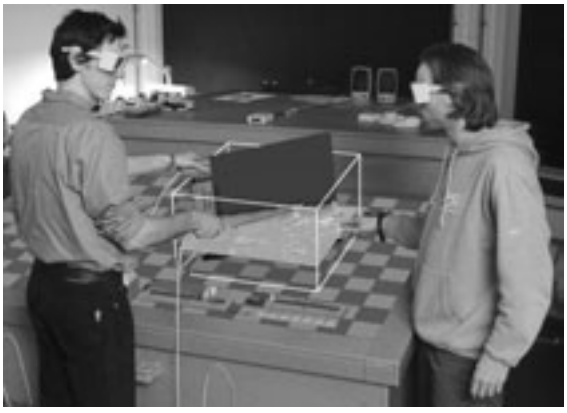


Figure 5.1: Two-handed exploration tools in data visualization on the Virtual Workbench (see also the Color Section)

5.1.1 Introduction

Scientists use a visualization system as a tool in an effort to explore and interpret available data from their experiments, measurements or simulations. The

system must provide its users with means to effectively perform the data exploration task. The interactive nature of this task implies that the man-machine interface plays a prominent role in designing effective visualization applications. VR has a potential to enhance this two-way communication between the researcher and the visualization application. In contrast to desktop-based 3D visualization applications, the use of stereoscopic, (semi-)immersive displays and spatial interaction devices provides the user with a 3D experience of datasets, allowing a quicker exploration.

Although VR can provide us with an intensified view of data, and more intuitive ways of interaction with the environment, some problems concerning data exploration tasks have not been solved. These problems include high interactivity of the visualization system and the virtual environment, effective navigation in the visual representation of the data and good control of visualization tools and their parameters. The datasets from simulations and measurements from various research areas are multi-modal and multi-dimensional, growing rapidly in size and dimensionality. Advanced visualization techniques such as direct volume rendering or iso-surfaces are computationally intensive and their performance degrades dramatically as the dataset size grows. In addition to these performance implications, the interactive control of the input parameters (e.g. selecting an iso-value or defining a transfer function) to achieve useful graphical representations of the data is a complex task. Instead of concentrating on the acceleration of visualization techniques we focus on the exploration process itself. We developed intuitive interaction scenarios to support interest driven exploration. Intuitive navigation and the use of simple and fast interactive visualization tools provide a useful approach to the effective exploration of volumetric data.

We will first give an overview of two-handed interaction techniques in visualization applications in VEs. We define appropriate interaction scenarios using both the Plexipad and the stylus, which form the basis for our two-handed interaction tools on the Responsive Workbench. Based on these interaction scenarios, we describe our set of implemented exploration tools and their characteristics. We focus our description on navigation and probing tools. After an overview of implementation details, example visualization applications from different research fields will demonstrate the use of our tools in practice.

Related Work

A good overview of the various challenges of scientific visualization in VR is given in [LaViola, 2000]. One of the main challenges described in this report is to "make interaction comfortable, fast, and effective". In this section we concentrate on this topic, focusing on the intuitivity of the interaction techniques involved in the exploration and visualization of data. Based on promising results of working with both input devices simultaneously, we have focused on the use of two-handed interaction techniques.

Numerous studies have shown that Guiard's framework [Guiard, 1987] is useful as a guideline for designing a two-handed interface. His findings in the distribution of labor between two hands in everyday activities not only proved useful for 2D computer drawing interaction schemes [Kabbash *et al.*, 1994; Bier *et al.*, 1993], but was also applicable to the study of two-handed interface scenarios in VEs [Schmalstieg *et al.*, 1999; Cutler *et al.*, 1997; Szalavari & Gervautz, 1997; Hinckley *et al.*, 1994; Lindeman *et al.*, 1999; Coquillart & Wesche, 1999]. The following types of interaction tasks can be distinguished:

- **One-handed task:** only one hand performs a task.
- **Double one-handed task:** each hand performs a separate task.
- **Two-handed task:** both hands co-operate to perform a single task.

The division of tasks between hands in the case of two-handed tasks can be either symmetric or asymmetric. A two-handed task is symmetric when both hands perform identical actions. In an asymmetric task, the most common form of two-handed tasks, each hand performs an individual action, involving a complex coordination between hands. The dominant hand is the preferred hand for precise movements such as writing. For most people this is the right hand; the non-dominant hand provides guidance and support. For asymmetric two-handed tasks, Guiard described the following principles:

- **Dominant to non-dominant reference:** the motion of the dominant hand finds its spatial reference in the results of the motion of the non-dominant hand.
- **Asymmetric scales:** the right and left hand are involved in different motions. The motions of the non-dominant hand tend to be of lower frequency and higher spatial amplitude. In other words, the non-dominant hand is responsible for the infrequent large motions while the dominant hand has more movements in a smaller area.
- **Non-dominant precedence:** the movement of the non-dominant hand precedes the dominant hand. The dominant hand waits for the non-dominant hand to initiate and set the spatial reference before engaging action.

Most of the work in the field of two-handed interaction interfaces concentrates on object manipulation or assembly tasks [Cutler *et al.*, 1997]. In addition, most reports on two-handed interaction in VR deal with two identical input devices for each hand, such as wands or gloves. Similar approaches that use a hand-held panel and a stylus, are described in [Szalavari & Gervautz, 1997; Lindeman *et al.*, 1999; Coquillart & Wesche, 1999]. The use of a transparent panel on a projection based table has been reported in [Schmalstieg *et al.*, 1999], but the panel is not actively used in a visualization process. Alternative approaches to the improvement of man-machine interaction use other input modalities such as speech and gesture recognition.

An example of this multi-modal interface in a visualization application is described in [LaViola, 2000]. We use the two-handed task principles to match the interaction scenarios for the tools in our system. These scenarios are reflected by various interaction tools for navigation and probing.

5.1.2 Interaction Methods

The Responsive Workbench provides a VE on a laboratory table. The user stands in the real world and looks down into the virtual world. Instead of bringing the user into the virtual world, the virtual world is brought to the user [van de Pol *et al.*, 1999]. The virtual workspace is usually within reach of the user's hands, but can also extend under the projection surface. Most of our tools are therefore designed to work both directly and remotely. The projection surface of the RWB provides passive haptic feedback and is suitable for the placement of 2D/3D interaction widgets. In our VR setup we use the following two input devices (Figure 5.2) which are tracked by electromagnetic trackers with six degrees of freedom (DOF): position and orientation.

- **Stylus:** a pen-shaped input device with a single button. The pen can define a point in space (zero-dimensional or 0D). The shape of the pen defines a directional reference axis. The pen can be used to intuitively specify a line in 3D space, extending the actions from 0D to 1D. This function is often used by ray-casting selection.
- **Plexipad:** a lightweight transparent acrylic panel (300 x 300 x 2mm) on which a tracker sensor is mounted. The pad can be used to naturally position and orient a 2D plane. It defines a 2D reference plane in 3D space. In contrast to a similar prop presented in [Schmalstieg *et al.*, 1999], the tracker sensor is mounted under a foam handle at the edge of the panel. The handle allows a firm and comfortable palm grip, reducing fatigue in the fingers. With the tracker mounted close to the wrist the inconvenience of the tracker cable (obstruction of the view, weight on the panel) is reduced to a minimum.

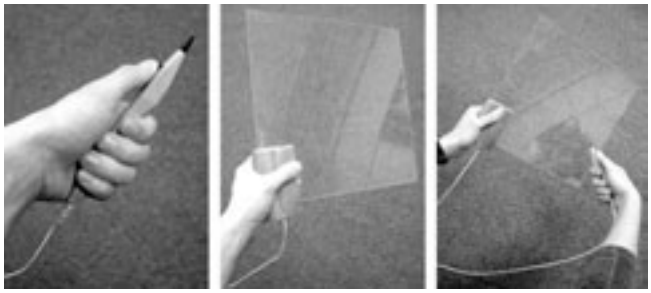


Figure 5.2: The stylus, the Plexipad and their two-handed use

In our two-handed interaction setup, the dominant hand holds the stylus while the non-dominant hand holds the Plexipad. The stylus and Plexipad are interchangeable, and this allows both right-handed and left-handed persons to operate the tools.

Interaction Scenarios

Based on our input devices and Guiard's principles of two-handed tasks we have derived the following interaction scenarios:

One-handed interaction: either the stylus or the Plexipad is actively used to interact with the environment. The stylus is suitable for direct (0D) or ray-casting (1D) selection and manipulation, where the Plexipad allows direct control (positioning and orientation) of objects which are virtually attached to the Plexipad.

Double one-handed interaction: the stylus and the Plexipad are used to perform unrelated one-handed tasks. The Plexipad and stylus each have their own separate functionality: a direct coupling between the tools is absent. This scenario allows a combination of stylus-based and Plexipad-based one-handed interaction scenarios. Although this direct relation between the two interaction tasks is absent, usually a higher-level goal will be pursued.

Symmetric two-handed interaction: Both hands perform identical tasks. This type of interaction task is not likely to be used in our scenarios, considering our use of two distinct input devices. Systems that use two identical input devices like wands or gloves usually support symmetric two-handed interaction [Cutler *et al.*, 1997].

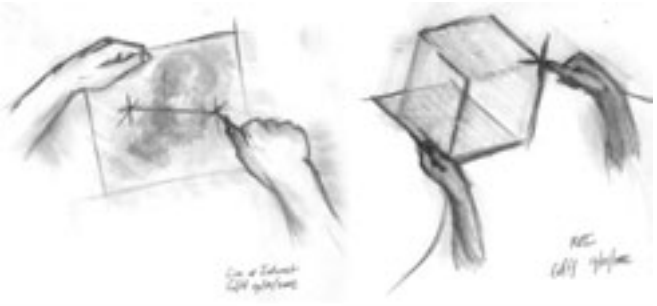


Figure 5.3: Asymmetric two-handed interaction: constrained actions (left) and complex actions (right)

Asymmetric two-handed interaction: the Plexipad sets the 2D reference plane for the stylus. This combination of the two tools exploits the familiarity with the "pen and pad" metaphor. In addition, the pad provides tactile feedback to the stylus' movements. The combination of the plane shaped panel and similarly shaped virtual object proves to be very intuitive. It feels as if you are holding

the virtual tool in your hand. We distinguish the following three scenarios for this asymmetric two-handed interaction:

- **The Plexipad serves as an object container.** The Plexipad is used as a container for 2D and 3D virtual objects or interaction widgets, which can be manipulated or operated by the stylus.
- **The Plexipad constrains the stylus.** The actions of the stylus are projected on the reference plane defined by the Plexipad. This actively constrains the 3D actions of the stylus in a 2D plane (Figure 5.3, left).
- **The Plexipad and stylus are used for a complex interaction task.** Here the Plexipad and stylus form a pair of input devices that control a single complex interaction task. Examples of this interaction scenario are the selection of a region of interest (Figure 5.3, right), a 3D lasso selection tool, a modeling tool or a cutting tool.

The Plexipad as a passive object container is widely used and is well suited for hand-held 2D or 3D menus and object snapping [Lindeman *et al.*, 1999; Schmalstieg *et al.*, 1999]. Although we also use the Plexipad for this purpose, we concentrate on more dynamic asymmetric two-handed interaction scenarios. The second scenario uses the Plexipad as a pure 2D reference plane for stylus interaction. In the third scenario however, the Plexipad actively participates in the interaction. This complex interaction has been used mainly with two 0D/1D input devices such as pens or gloves [Cutler *et al.*, 1997]. The plane shape of the Plexipad can be exploited to create more expressive interaction tools. We use the concept of holding two dimensions in one hand (the Plexipad) while the other hand (the stylus) controls the third dimension. In the following section we will describe how these interaction scenarios are reflected in the design of our interaction tools for navigation and probing.

Interactive Exploration Environment

We have implemented a VE for visualization and exploration of data on the RWB (Figure 5.4). Besides a conventional 3D GUI, the user can interact with various visualization tools and the data space which can contain both volumetric and object data. The data space is represented as a virtual object, using an outline to indicate the spatial boundaries.

In the description of the exploration tools we will use a Molecular Dynamics (MD) application, our testbed during development, to illustrate their functionality. MD is used to study the properties and behaviour of complex particle systems. This application is a good example of combining volumetric fields with object information (particles). The various illustrations in this section show the study of a solid electrolyte (sodium beta-alumina) which consists of a crystal molecular structure with layers of sodium ions. In Chapter 6.2 the MD application will be described in more detail.

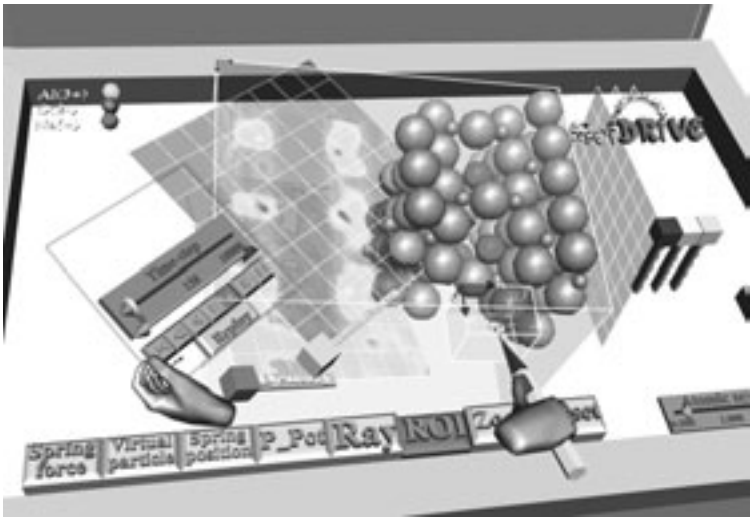


Figure 5.4: Overview of the VE for data exploration and visualization

A typical exploration process begins with a quick spatial and temporal scan of the volumetric data for interesting information. If an interesting region or phenomenon has been found in the data, the attention is focused on this aspect of the data. The user then tries to get detailed insight in this data by inspecting or probing the various data values in the neighborhood, using visualization techniques or measurement tools to explore the data.

In our approach we provide interaction tools to support these steps of the exploration process. These can be divided into two main categories: *navigation* (positioning, orienting, cropping and zooming of the data space) and *probing* (localized visualization and measurement of the data).

5.1.3 Navigation

In our VR concept we do not consider navigation as flying through a VE. Instead we use the laboratory table metaphor, where the position of the VE with respect to the physical table is fixed. Navigation consists of manipulation actions on the data space object which contains all data. Common direct or ray-cast manipulation tools can be used to position and orient the data space object. For other types of navigation we have developed the following interest driven tools.

Zoom Tool

The zoom tool allows the user to take a closer look at a point in the data. In naive implementations (e.g. using a slider to adjust the zoom factor), the user has to pay attention to both the manipulation of the slider and the size of the object. Moreover, the object will often scale from its local origin, thus effectively moving

the point of interest to another position. We observed users repeatedly zooming and repositioning in an effort to get a good view of the point of interest in the object. Our "magnifying glass" metaphor is interest driven: click at a point of interest, pull back to zoom, push away to zoom out.

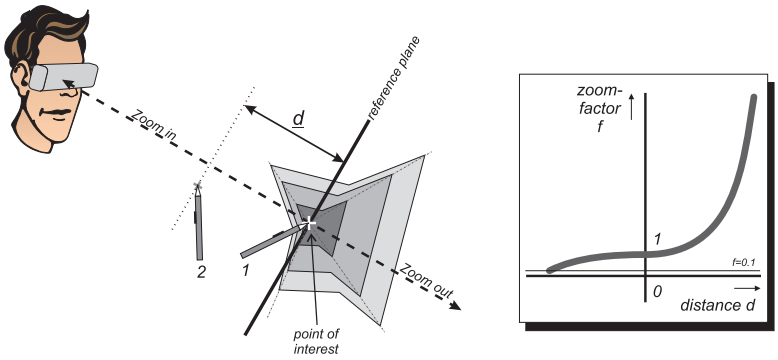


Figure 5.5: Zoom tool and scaling function $f(d)$

After activating the zoom tool, the user can specify the point of interest in the VE (Figure 5.5, stylus position 1). The zooming will occur around this focal point, so that this point will remain stationary. The line from this point to the eyes defines the normal of the reference plane. The user can move the stylus away from the initial position to adjust the zoom factor. If the stylus is moved towards or away from the eyes, the zoom factor increases or decreases respectively. This factor is determined by distance d, the perpendicular distance from the stylus (position 2) to the reference plane. To prevent jerky zooming with small movements while allowing large zooming with large movements, we use a non-linear scaling function $f(d)$. In addition the zoom factor is clipped at $f(d) <= 0.1$. This prevents the disappearance of the zoomed object.

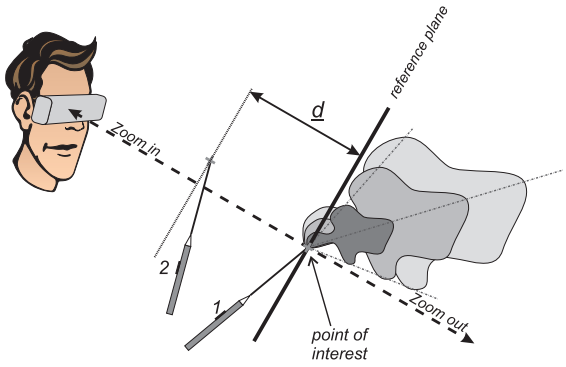


Figure 5.6: Ray-cast zoom tool for distant objects

Sometimes the user wants to zoom from a point that is out of reach (too far away or under the table). To facilitate this we have extended the zoom tool with the possibility to select a point of interest on a remote object using ray selection (Figure 5.6).

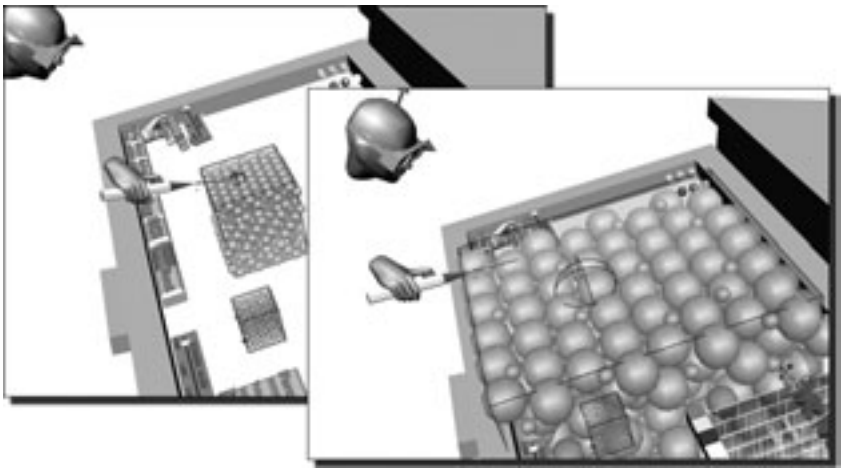


Figure 5.7: Ray-cast zoom tool used in molecular visualization (MolDRIVE)

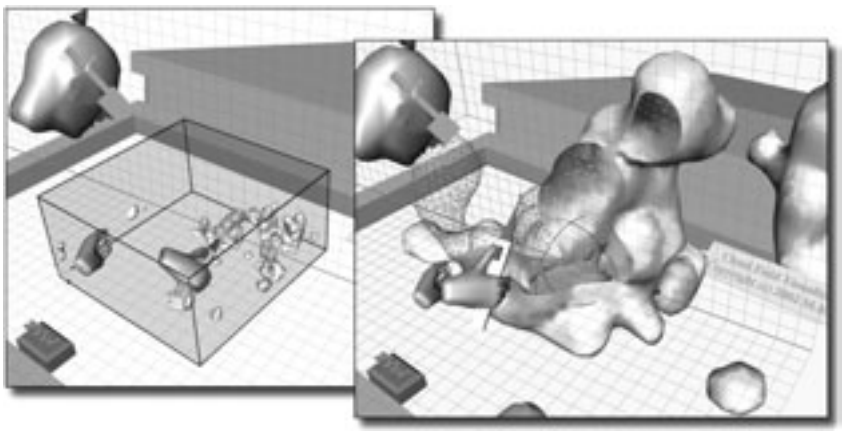


Figure 5.8: Direct zoom tool used in cloud visualization

The zoom tool using the "magnifying glass" metaphor has been found very intuitive, easy to learn and useful by users, see Figures 5.7 and 5.8. It has significantly decreased the time and effort needed for obtaining a more detailed view on a point in space.

Mini System tool

Using the zoom tool the user can be fully immersed in the data space/domain, thus losing orientation. The mini system provides a global context of the data space in the form of a small model (Figure 5.9). To have a good navigation control in the data space we coupled the orientation of the mini system to the data space object. If the user rotates the data space object, the mini system is rotated accordingly. Likewise, if the user rotates the mini system, the data space object is rotated as well. As a result the system and its miniature are always aligned.

The data space object is rotated around a point of interest, for which we selected the center of the projection screen. We decided not to share positioning information between the data space and its small version. A small repositioning of the mini system would cause a much greater repositioning of the data space object, thereby confusing the user.

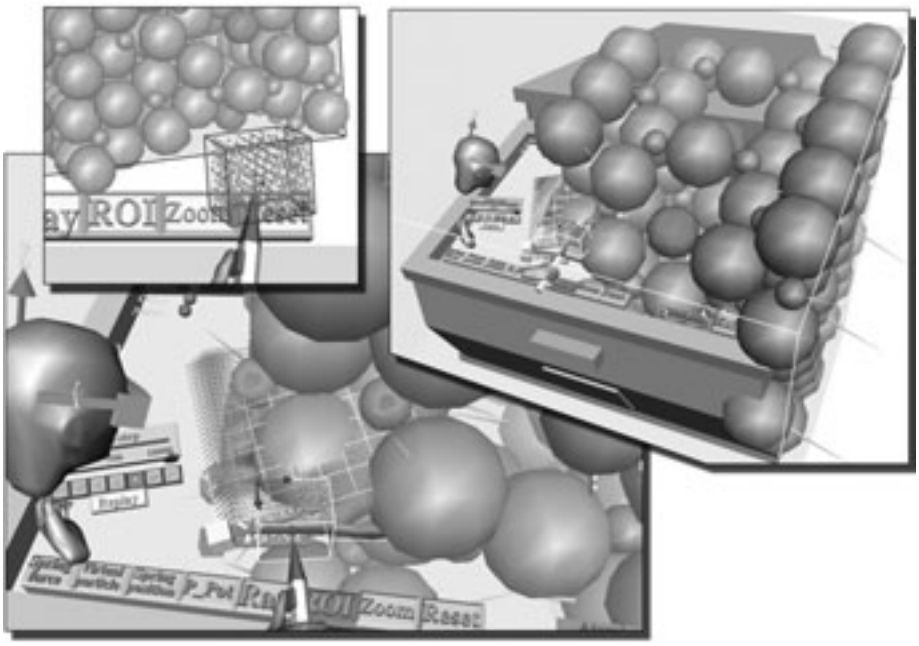


Figure 5.9: Mini System tool helps to navigate in molecular visualization (MolDRIVE)

An advantage of not using the location information of the system is that the mini system can be placed anywhere in the VE. For example, the user can use the mini system to rotate the data space object and then move it out of sight, e.g. to the side of the screen.

Region of Interest tool

The region of interest (ROI) interaction tool allows users to select an arbitrarily oriented 3D box in the VE. The stylus is used to define two points in space to define an extent of the box. This box is created by clicking on the surface of the Plexipad. While holding the button, the user can move both stylus and Plexipad to dynamically adjust the size, position and orientation of the box.

The Plexipad also defines the orientation of the base plane of the box. This enables the user not only to reposition the box but also to adjust the orientation of the box dynamically. The complex asymmetric interaction between the two hands reduces the number of actions, allowing easy creation of axis-aligned boxes as well as arbitrarily oriented boxes in 3D space in a single movement.

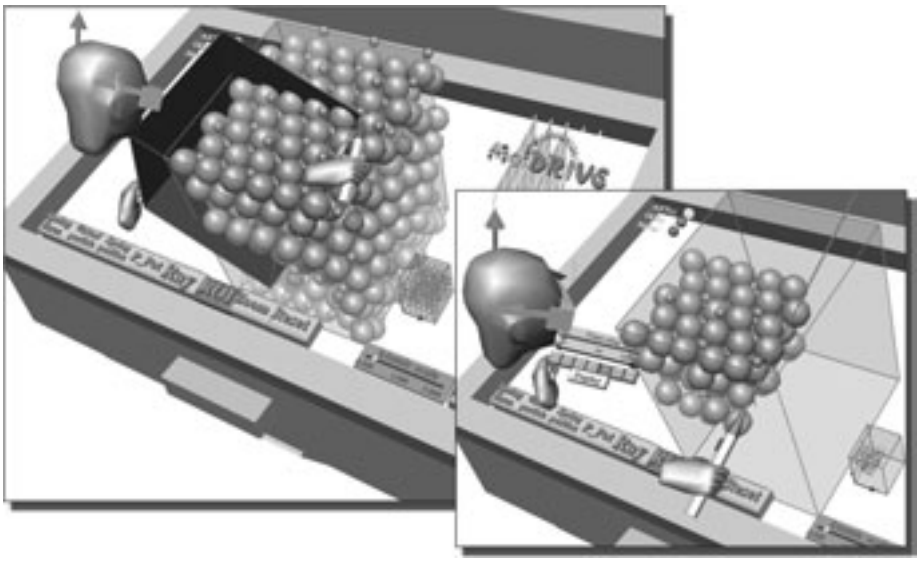


Figure 5.10: Region of Interest tool in molecular visualization (MolDRIVE)

In our MD application we use the box to define the 3D region in which we want to display particles (Figure 5.10). The technique can also be used for object selection, volume probing or 3D modeling. The Plexipad can control the 2D base of an object while the position of the stylus defines another parameter such as height or extent.

5.1.4 Probing Tools

The probing tools, which allow the user to inspect the data, work in synergy with the navigation techniques. Probing in a dataset is a necessary input for various visualization and measurement tools. 3D probes such as point probes (0D), line probes (1D), plane probes (2D) and volume probes (3D) are therefore essential in the data exploration process. Without an appropriate feedback it is difficult to freely probe the data in 3D space. The passive haptic feedback provided by the Plexipad is very useful during 3D probing. We take advantage of the two-handed input scenarios defined earlier, using the Plexipad as a 2D reference plane. This reference plane forms the basis for the creation and positioning of probes.

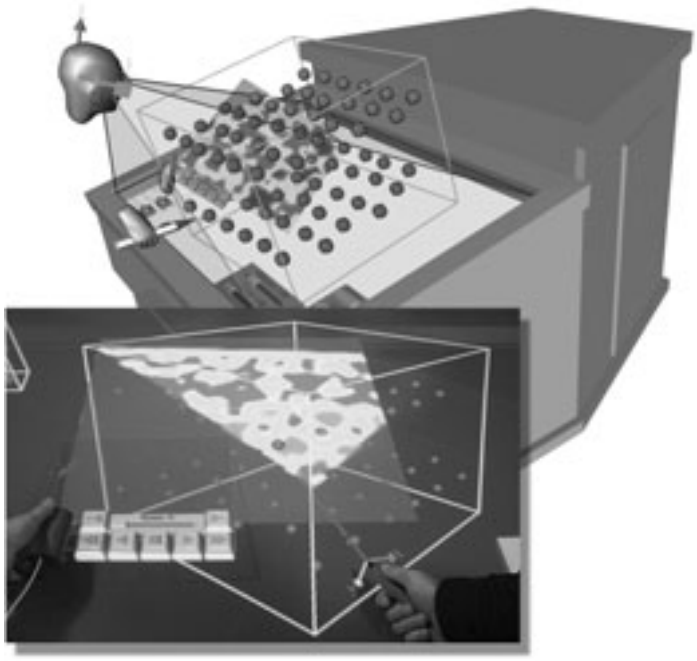
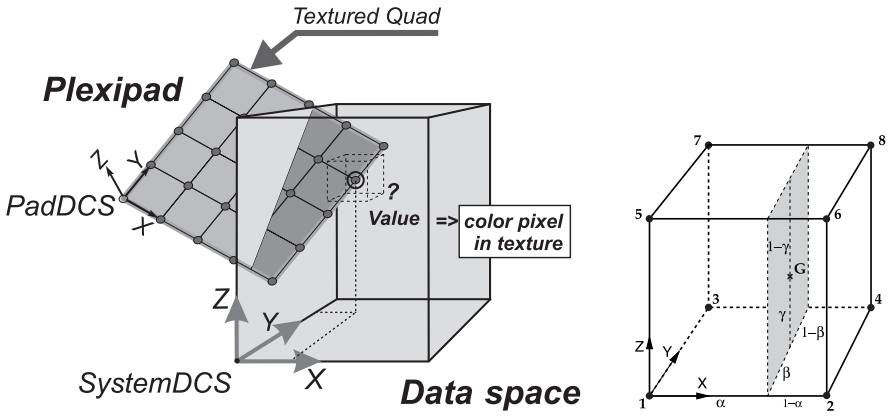


Figure 5.11: Direct data slicer: a user slices through an atomic density field of a solid electrolyte (MolDRIVE).

Plane Probing

The Plexipad allows the user to navigate a 2D plane through the VE. We exploit the interaction scenarios by attaching a plane-shaped probing tool directly to the Plexipad. This consists of a grid of point probes, which perform a trilinear interpolation of the volume data values, see Figure 5.12. We directly visualize these data values on the plane-shaped probe by using a textured rectangle (quad).



(a) (b)

Figure 5.12: (a) Direct data slicer is used to probe the volumetric data in a plane. (b) Trilinear interpolation inside a data cell.

Each pixel of the texture corresponds with the value at a point in an intersected cell of the data, which is calculated using a trilinear interpolation:

$$\begin{aligned} f_G \quad = \quad & f_1(1-\alpha)(1-\beta)(1-\gamma) + f_2\alpha(1-\beta)(1-\gamma) + \\ & + f_3(1-\alpha)\beta(1-\gamma) + f_4\alpha\beta(1-\gamma) + \\ & + f_5(1-\alpha)(1-\beta)\gamma + f_6\alpha(1-\beta)\gamma + \\ & + f_7(1-\alpha)\beta\gamma + f_8\alpha\beta\gamma \end{aligned}$$

where α, β and γ are relative positions of point G in the cell, with $0 \leq \alpha, \beta, \gamma \leq 1$.

As a result, the user directly controls the position and orientation of the probing tool, slicing through the volumetric data. We call this tool *the direct data slicer*, see Figure 5.11. The user holds the direct data slicer in the non-dominant hand and can quickly probe through the volume to get an overview of the data values inside. At the same time, the dominant hand can be used to operate the stylus for manipulation, zooming of the data, and the selection of new tools.

The direct data slicer forms a two-dimensional reference plane for other tools: for asymmetric two-handed interaction and passive haptic feedback, and a slicing plane in a volumetric dataset. This combined feedback assists in selection of points and lines in 3D space by using the stylus directly on the Plexipad. This assisted or constrained selection can be effectively used to define the input for visualization or measurement tools. The advantage of the plane probe is that it can also serve as a reference plane for other probing tools.

Point Probing

The point probe allows the user to request the data value(s) at a point in 3D space using the stylus. In addition to freehand probing, the direct data slicer can provide a tactile feedback for constrained probing, by clicking with the stylus on a point of interest on the Plexipad (a 0D selection on a 2D slice). An example of using a point probe is the *click iso-surface tool*, see Figure 5.13. This tool uses the value of an interactively positioned point probe as the iso-value which is used to create an iso-surface instantly. The iso-surface is generated by the marching cubes module in VTK [Schroeder *et al.*, 1999] and transferred to the VE using vtkActorToPF [Web-VTK-to-Perf], see Section 5.2.4.

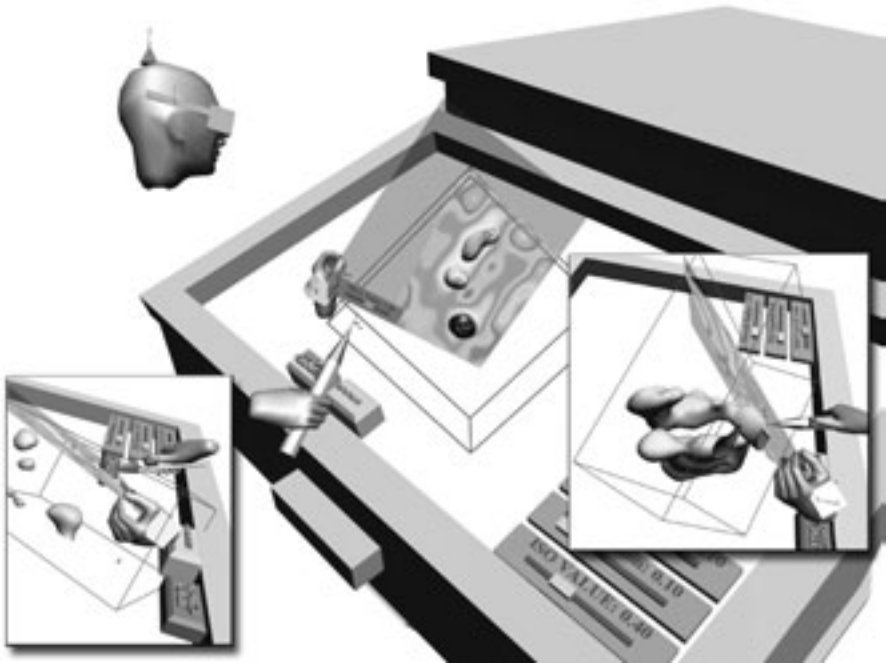


Figure 5.13: Point probing: the data value at the selected point is used as input for the iso-surface visualization tool (see also the Color Section).

Line Probing

Line probing provides a way of selecting a line in 3D space. Again, the direct data slicer can be used to create a 2D reference plane in 3D space, allowing the user to draw a line on the Plexipad (a 1D selection on a 2D slice). An example of line probing is using the probed data values along the selected line to define the data range of a color mapper. The color mapper defines the color-coding of the values probed by the direct data slicer. The line probe can be used to select

a line of interest (LOI) after which the color mapper is calibrated to reveal small variations in the selected profile, see Figure 5.14. This profile can also be used to select an appropriate iso-value for an iso-surface.

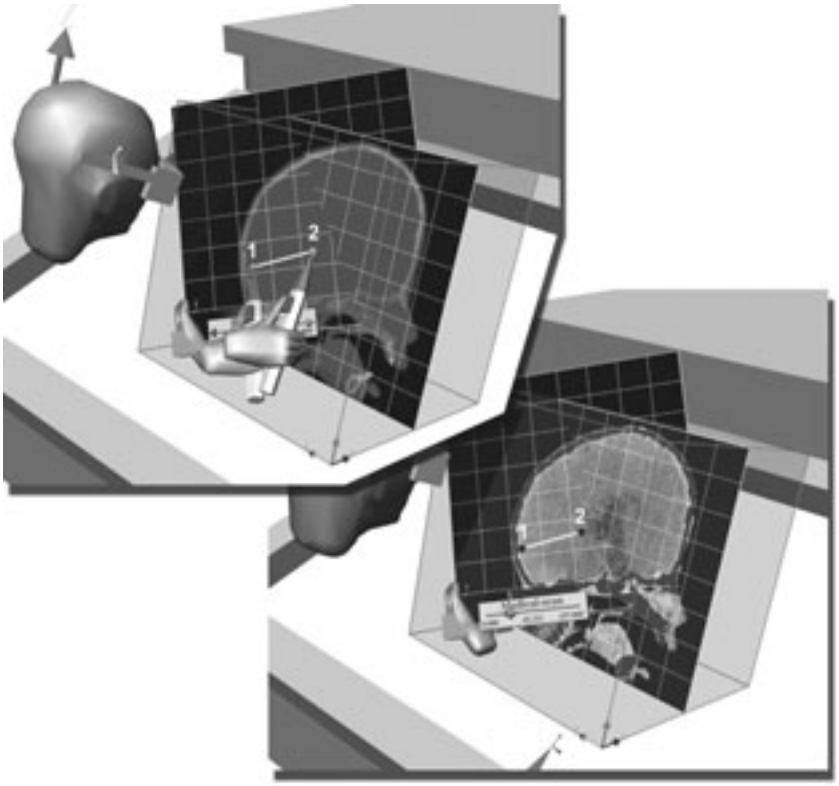


Figure 5.14: Line probing: the user selects a line of interest to calibrate the gray-scale color mapper in the visualization of a medical CT-scan.

Sub-Volume Probing

The principle of the oriented 3D box described in ROI (Section 5.1.3) can be used for probing sub-volumes. The user can crop or cut a selected volume of the data, while data slicers probe and visualize the data on the inside faces of the 3D box. The box selection can also be used for direct volume rendering in the given region. It is important that the user can interactively select and adjust the box while visualizing the data at the same time. After the initial exploration process, the presented probing tools can be used for interactive quantitative measurement purposes by presenting the probe results in a classic 2D or 3D graph.

For example, the data values along the LOI can be presented in a 2D graph to visualize the data profile along this line. Another example is to display the results of a point or line probe over a period of time. These dynamic graphs can be placed anywhere in the VE and stored for further analysis. We expect that this data representation will be appreciated by researchers for its analytical and quantitative character.

5.1.5 Implementation

The interaction tools presented were implemented in the VRX toolkit (Section 5.2) on top of the RWB Library (Section 3.4).

The RWB Library uses a default *RWB-Interactor* class, which can select and manipulate objects directly with the stylus or using ray-casting for distant objects. It checks for bounding volume or ray-casting selections, providing the intersection and interaction points (stylus/ray with objects). This class works on the principle of an interaction finite state automaton, driven by interaction events. RWB-Objects can be selected and deselected, picked (first button click), manipulated and released, each time invoking their event callback function. This way the desired behaviour of virtual objects can be implemented.

Using the generic RWB-Interactor class we can define new interaction tools, such as the ZOOM and the ROI or the Spring Manipulator [Koutek *et al.*, 2002]. We have implemented two interactor schemes.

The user extension of the RWB-Interactor executes the default RWB-Interactor first to perform the intersection checks and to obtain a new interaction state and interaction points; then the user interactor function performs the desired behaviour *(i.e. ZOOM, ROI, Click-ISO, Probing Tools)*

The fully user-defined RWB-Interactor overloads the functionality of the default interactor, processes the input sensors itself (via RWB-Lib), and implements the desired functionality *(i.e. Spring Manipulators, Virtual Particle Steering)*

The user RWB-Interactors can be activated by clicking on a button widget. The RWB-Interactor can be de-activated by the tool itself or by activating another interactor. For example the zoom tool is automatically deactivated after completing a zoom action and control is returned to the default RWB-Interactor. The design of the interactor determines the way the tool works. The event-based interactors, straightforward coordinate transformations between the Plexipad and stylus and the simulator environment all provided by the RWB Library allow rapid development of new (two-handed) interaction tools and scenarios.

5.1.6 Example Applications

We will illustrate the interaction scenarios described above with two applications. Visualization and steering of MD simulations have been an important

inspiration for our research. Within the scope of MD visualization research we have developed most of the interaction techniques presented. To prove their wider applicability we also applied them in other case studies.

Particle Steering in Molecular Dynamics

In our VR lab we have developed the MolDRIVE system [van Hees & den Hertog, 2002; de Haan, 2002], a VR system for visualization and steering of real-time remotely running MD simulations, see Section 4.3. As described earlier, MD simulations are used to study properties and behaviour of particle systems.

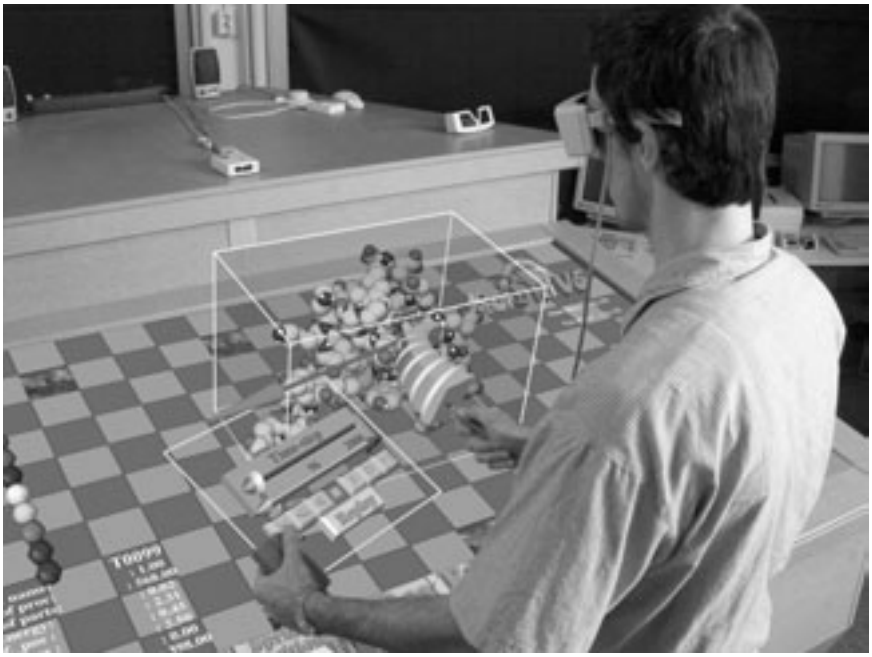


Figure 5.15: Molecular Dynamics: user performs particle steering with Spring Manipulator while holding the time-control of the remotely running simulation in the left hand (see also the Color Section).

In the MolDRIVE system we work with particle data (positions, force and velocity vectors) and volumetric data. The volumetric data consists of regularly structured grids of scalar data (e.g. kinetic energy, potential energy, particle density) and vector data (e.g. force fields). As we are dealing with a real-time simulation, the VE content is updated each time we receive new data. We have used double-buffer data management so that the update of visualization tools is not disturbed when the simulation delivers new data, see Section 6.2 for more information.

Growing interest for the ability to steer particles in running simulations has led us to develop particle steering tools. The most reliable steering is provided by the Spring Force Manipulator (Section 4.3), which has been derived from the Spring Manipulation Tools, providing visual force feedback during manipulation, see Figure 5.15.

The task of particle steering can be effectively assisted by the direct data slicer. A higher level interaction is achieved when for example the stylus is used for dragging a particle, while the Plexipad is used to gain information on the potential energy around the particle, see Figure 5.16. In MD simulations the particles usually have preference to move from higher to lower potential. Using the direct data slicer the user can see what the most efficient trajectory for the particle will be and then uses the steering tool (Spring Manipulator) to drag the particle in that direction. The interaction of the non-dominant hand itself does not influence the reference frame of the stylus.

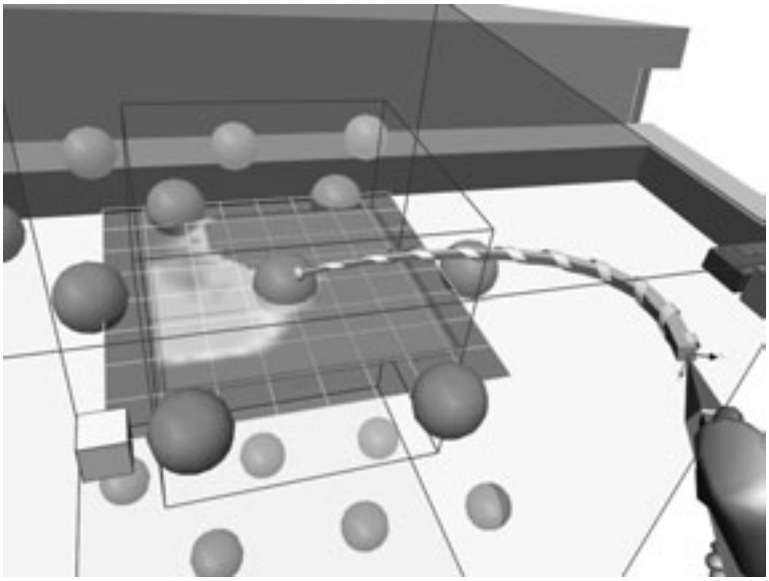


Figure 5.16: Molecular Dynamics: the direct data slicer is used to display potential energy around an individual atom during particle steering with the stylus.

Instead it presents valuable information which can be interpreted to adjust actions with the dominant hand. The visualization tool in the non-dominant hand provides information that allows the user to redirect simulation steering actions with the dominant hand.

Visualization of Cloud Simulations

Atmospheric simulations are usually very complex and computationally intensive, and they produce large time-dependent datasets. In this case study we are dealing with data originating from Large Eddy Simulation (LES) with cumulus clouds. Scientists study atmospheric boundary layers to get a better understanding of the turbulent dynamics and the behaviour of clouds (cumulus, stratocumulus). Turbulent convective motions are very important since they are responsible for the vertical transport of heat, moisture and pollutants. The presence of clouds in the boundary layer makes the dynamics even richer but forms an additional complication due to the phase changes (condensation / evaporation) [Siebesma & Jonker, 2000].

The LES in this case has produced a large dataset (20 GB), with grid dimensions of 128x128x80 and 600 time-steps, which means one hour of simulated clouds in an area of 6x6x3 km. The key quantities are momentum (velocity), pressure, temperature and moisture. It is a real challenge to be able to interactively visualize and browse through such a large time-dependent dataset. The exploring user is searching for an interesting cloud with a complete life cycle inside the simulated time interval. The spatial relations and the simulated physical properties around the selected cloud must be explored in detail.

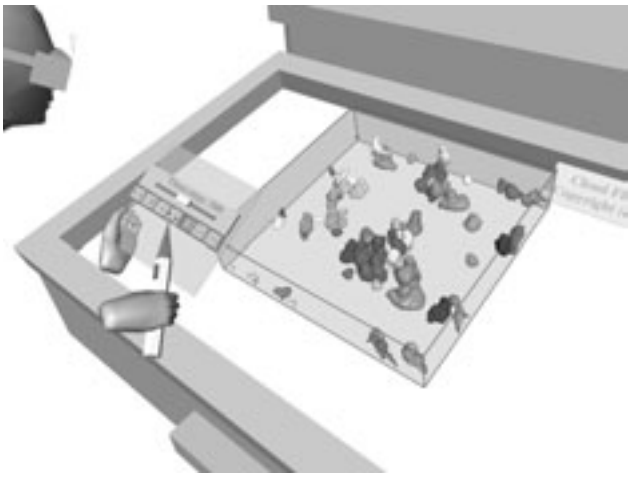


Figure 5.17: Cloud simulation: playback of the cloud field simulation. Clouds are tracked in time and colored accordingly. The user is searching for a cloud with interesting properties.

The presented two-handed interaction scenarios have also been applied in this case study. The exploration process begins with a quick search through the dataset. The Plexipad contains a time-control widget to navigate in time of the cloud simulation. The stylus can be used for operating it, as well as for navigation in the visualization of the cloud field, see Figure 5.17.

The direct data slicer can be attached to the Plexipad as well, enabling highly interactive exploration of the data, see Figure 5.18. The stylus can be used to probe the data on the surface of the direct data slicer. While using the direct data slicer, the stylus is used to operate the rest of the VE, using the 3D GUI to change for example the visualized data or adjusting the color mapper. A color mapper widget can also be attached to the data slicer, enabling the color mapping adjustments without changing view context. The LOI tool can be used to calibrate the color mapper on the selected data range.

Figure 5.18: Atmospheric visualization: the direct vector slicer shows the flow momentum around the cumulus clouds (iso-surfaces of liquid water concentration).

5.1.7 Results

We have presented the interaction tools and a new way of exploring data in 3D to the scientists whose data we visualize and they became quickly familiar with this exploration interface. We have asked them what makes this concept that easy to use and intuitive. It is exactly the pen and pad that form a natural interaction pair. Our conclusion at this point is that intuitive data visualization and exploration tools for VR should relate to real tools and interaction paradigms, which are used in the real world when scientists make their observations. Although we give them a slightly different look and functionality, people will have no problems using them. Another aspect is that people are working in 3D in the real world. When performing tasks like drawing, construction or measuring, they usually search for a supporting plane or a reference. Measuring freely in the air is difficult. The passive haptic feedback and constraints are important.

In our solution it is provided by the transparent Plexipad, which is augmented by virtual objects and tools with straightforward meaning.

5.1.8 Intuitive Exploration Tools: Summary and Discussion

We presented intuitive interaction scenarios for the Plexipad and the stylus, based on well-founded two-handed interaction paradigms. The described interaction scenarios provide a usable approach to the development of navigation and probing tools. The use of two-handed scenarios proved valuable for the development of volumetric visualization tools in projection-based semi-immersive VEs, like the Responsive Workbench. The complex two-handed interaction scenario, holding the two dimensions of the Plexipad in one hand and controlling the third dimension by the stylus in the other, allows the creation of complex but still easy-to-use interaction tools. This two-handed synergy between the stylus and the Plexipad allowed natural exploration and probing of volumetric datasets.

The combination of the zoom tool, the region of interest tool and the mini system allows a flexible way of navigating through the data and focusing on interesting regions of the data while maintaining a view of the context. The direct data slicer presented can be used to quickly probe large datasets and scan for interesting phenomena in the 3D data space during navigation. The tactile and visual feedback provided by this tool also provide a spatial reference plane for various 3D probing tools, allowing accurate placement of measurement tools. Moreover, the presented interaction tools are not computationally intensive and their performance is independent of the data size.

A successful employment of the interaction concepts presented needs accurate tracking. The coherence of the Plexipad and the virtual object has to be optimal to achieve a good tactile feedback, which in turn increases the interaction experience when using the stylus on the Plexipad. As we are using electromagnetic trackers, we are dealing with tracking errors especially in the orientation of the Plexipad. Therefore a specialized tracker calibration scheme is needed, see Section 3.2.2. Current trends show a good alternative of using optical tracking.

Currently we employ the interaction techniques on various visualization applications, of which some are shown as examples in the case studies (Chapter 6). We can extend our concept with 3D measurement tools, and also concentrate on exploration of large multi-modal and multi-dimensional datasets. The use of our probing techniques in combination with complex time-critical visualization tools will allow us to explore and analyze large and complex datasets more effectively.

5.2 VRX: Virtual Reality eXplorer

5.2.1 Overview of the Concept

Concurrently with the development of the RWB Library we have worked on the design and implementation of a modular object-oriented toolkit for exploratory data visualization on the Virtual Workbench. Our efforts have resulted in the Virtual Reality eXplorer (VRX), see Figure 3.14. We have designed a set of simple classes to support the development of visualization applications on top of the RWB Library, providing the multiprocessing support for visualization tools.

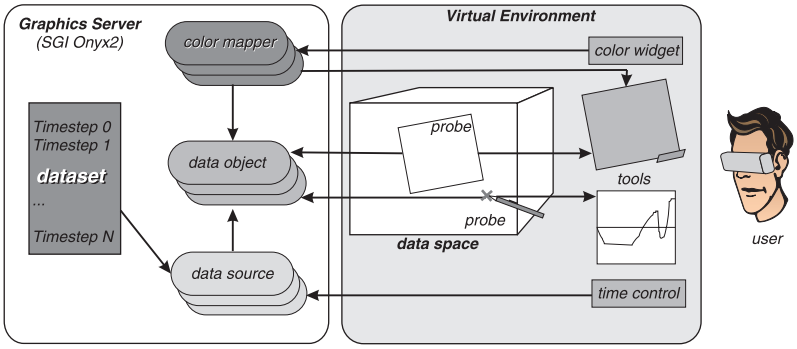


Figure 5.19: Schematic overview of the various modules in VRX

With the VRX toolkit we provide a framework for the development of applications for interactive visualization of multi-dimensio-nal and time-dependent datasets in VR. We focus not only on the actual interactive exploration and analysis of the data in VR, but also on the construction of the visualization pipeline. Similar to other existing visualization systems each class represents a modular function in the visualization pipeline. Instances of these classes can be interconnected to define the flow of data. The advantage of this modular approach is that each component in the set can be extended, optimized and tested separately without great implications for the other components. Once the modular framework and the communication is defined, developers create their own application specific modules for use in the system or the modules can be shared between several applications. We have created a basic application framework and some virtual tools to demonstrate our concept. We will describe now the main modules and their interconnections, see Figure 5.19.

Data source is responsible for the physical data file access and memory storage of the time-dependent datasets. At this point, the data source class is able to read and convert different file formats that contain regular grids of scalar or vector data. The structure of the filenames of the data files must be configured, while the data are stored per time step usually in separate files. We use an

asynchronous process for disk operations in combination with double buffering and custom semaphores not to slow down the graphics related tasks.

Data object represents one data entity (i.e. one scalar or one vector quantity) in the data space. It is an abstract layer on top of the actual data source and is responsible for handling data value requests by virtual visualization tools. The data object class presents an interface to those tools for calculating data values at both arbitrary points in 3D space, or at points on the data grid. If an arbitrary point in space is requested, it first calculates whether this point falls within the data space boundaries. If the requested point is outside the data, it returns a NULL. Otherwise, it calculates the cell in which the point is located and retrieves the value from the underlying data source. For this the data object uses tri-linear interpolation.

Data space is a box-shaped virtual object (RWB-Object) that defines the spatial extent of the data in the VE. The actual presence of data in the VRX system is provided by data objects, which are placed in the data space. Each data object is responsible for one data source. By connecting multiple data objects, multiple types of data can be present simultaneously in one data space. Navigation tools operate on the data space and can be used for navigation or spatial filtering. Visualization tools work inside the data space.

Virtual tools: By using interactive virtual tools a user can probe various data types present in the data space for exploration, visualization and measurement. These probe tools can be controlled by the stylus and/or the Plexipad. The tools do not have direct access to the data in memory, but use data objects to request values at a point in 3D space. Each virtual tool operates on one data entity, thus one data object. When a visualization tool is used, the received value can be mapped to a color by using a color mapper object. Bellow we will describe various interactive probing tools that we have implemented.

Abstract tools: In addition to these spatial tools we have also created abstract tools such as the *color mapper widget* and the *time control widget*. These widgets can be connected to the color mapper and the data sources, and provide direct interactive control of these abstract parameters of the visualization pipeline.

Although the design and implementation of the basic concept itself took a substantial amount of time and effort, this paid off in increase of code stability, readability, flexibility, and development time of visualization applications for the Responsive Workbench. By using an object-oriented design, new application specific tools can be derived and extended relatively easily from the basic classes and tools provided. Also, the abstract layer provided by the data object offers an effective interface to the data for new visualization techniques. VRX also contains improved 3D interaction and exploration techniques, as described in Section 5.1. We have applied the VRX concept on several case studies, see Chapter 6.

Further, we have also implemented an interface to VTK (Visualization Tool Kit) and incorporated it in VRX. A promising idea is to combine VRX with an existing visualization package, which uses the network builder paradigm, such as OpenDX. In this way, scientists could more easily prepare their visualization sessions on regular desktop computers.

5.2.2 Multiprocessing Scheme

As described in Section 3.4.2, the RWB Library initiates and executes the APP, CULL, DRAW and COMPUTE processes. Multiprocessor support using a separate parallel COMPUTE process in combination with *pfFlux* nodes has the potential to enhance performance of complex visualization tools.

The update of the visualization/exploration VRX tools is done in the COMPUTE process. This process runs in parallel on the fourth processor of the Onyx2, not interfering with the normal rendering pipeline. Within this process, each visualization tool is updated in a separate thread sub-process (Figure 5.20). In this way, the separate tools are updated in parallel and the slow tools do not affect the fast tools directly. When the update of a tool is complete, the new data in the pfFlux buffer is immediately available to the DRAW traversal stage in the following frame. This way the latency of the visualization tools is reduced, compared to sequential execution of tools' updates.

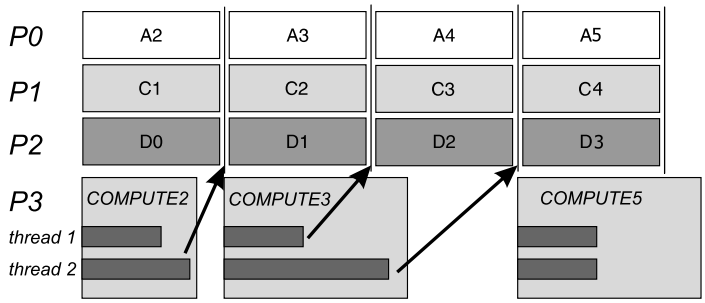


Figure 5.20: Multiprocessing in VRX. $P_{0,1,2,3}$ means processors nr. 0,1,2,3. A_i means APP traversal process of frame i. C_i, D_i stays for CULL and DRAW traversal processes. The COMPUTE process on proc. P_3 executes individual threads for each visualization tool. The arrows show a moment when the graphics data of updated tool will be passed to the DRAW process and displayed on the RWB.

5.2.3 Visualization of Volumetric Data

VRX mainly utilizes the idea of interactive exploration in VEs. The set of virtual visualization/exploration tools has been chosen carefully with a special attention to the level of interactivity of each tool.

In regular visualization applications, cutting planes (or data slicers) are often used to visualize slices of volumetric data. A data slicer probes the values in the data space at the intersection of the slicing plane and the volume. By mapping the data values to colored pixels in a texture, these values can be visualized in detail, see Figure 5.21. The number of sample points in the slicing plane can be specified by the user.

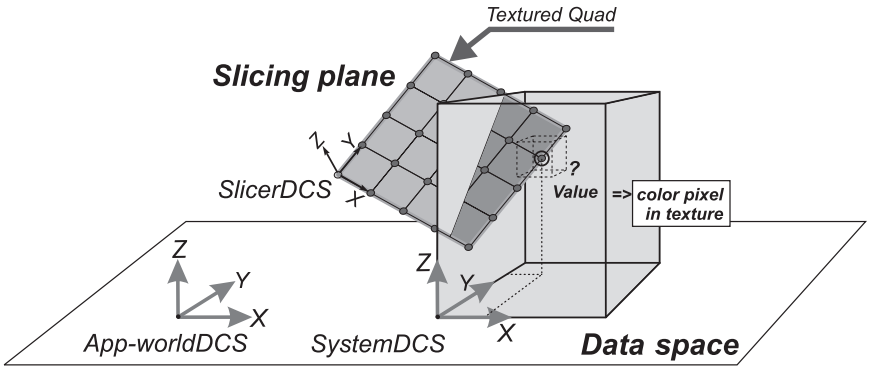


Figure 5.21: Data slicer is used to probe the volumetric data in a slicing plane.

We have developed and extended the concept of a generic data slicing tool. Together with other object-oriented classes of VRX we have designed a master class of a generic data slicer. Several data-specific slicers have been derived from this master class. These tools make data requests via abstract data objects. This way we could easily implement data slicers for scalar and vector data.

The functionality of the data slicer depends on the data space object and the slicing object. As described earlier, the data space object is a box-shaped virtual object containing a multi-dimensional dataset. The slicer object is a square, plane virtual object that defines the position and orientation of the slicing plane.

As a natural form of interaction we have designed and built the Plexipad, see Figure 5.2. The slicer object can be attached to the plane of the Plexipad. The sub-scene graph of SlicerDCS is then attached to a parent PlexipadDCS, which is updated with the tracker information. This way the slicer object can be freely moved through the virtual environment. The two-handed interaction scenarios with the Plexipad are described in detail in Section 5.1. The slicer object can also be attached to any other DCS of the scene. It is practical to attach the slicer objects under the SystemDCS of the data space. When the data space is manipulated (translated, rotated or zoomed), then the child objects are transformed in the same way, see Figure 5.23. It is also possible not to connect the slicerDCS with the SystemDCS. Then the slicer has to be a child object of the application world DCS (App-WorldDCS), which is also useful.

The slicer object contains a 2D grid of probing points. The grid dimensions are user-defined. A common value is 128x128 points for scalar data slicers and 64x64 points for vector data slicers. Each of the grid points has to be transformed from the local coordinate system of the slicer (SlicerDCS) to the coordinate system of the data space. This is implemented via homogeneous matrix transformations. The data values can be obtained via the data object.

Texture Slicer

The mapping of scalar data into a color space seems relatively straightforward. The colors on the slicer object visualize the data values at the intersection with the data volume (Figure 5.21). For this purpose a regular 2D texture is used (i.e. texture slicer). The hardware acceleration of textures provided by the IR2 graphics board in the Onyx2 allows the use of high resolutions (e.g. 512*512) without much loss of performance. The update of the values in the texture however does require matrix calculations and tri-linear interpolation. Therefore it is the main bottleneck of this tool.

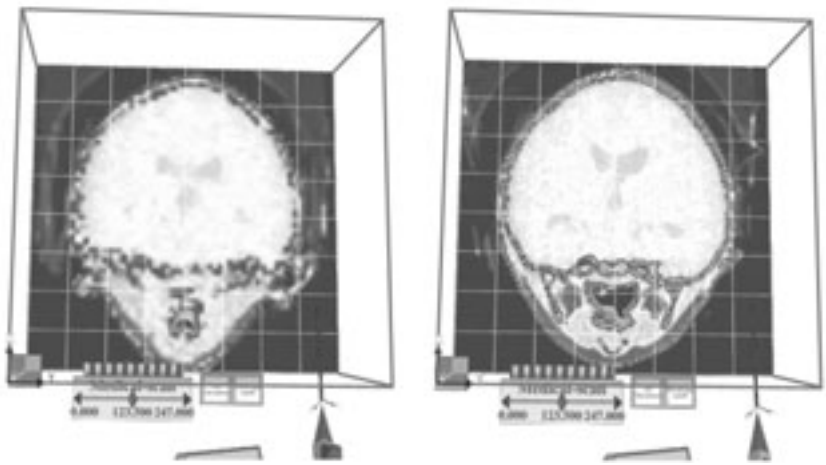


Figure 5.22: VRX - exploration of medical CT scan: adaptive resolution of the texture slicer; lower resolution during slicer manipulation (left) and high resolution when the slicer is not moving (right); the texture slicer is attached under the SystemDCS and is constrained to move in Z-direction only.

To reduce the influence of this bottleneck, an adaptive/variable grid resolution of the slicer can be used *(adaptive slicer resolution)*. There are three levels: high-res, mid-res and low-res (i.e. 128x128, 64x64, 16x16). When the user is manipulating the data slicer, a lower resolution is used. When the slicer remains still, the resolution is automatically set to high-res, see Figure 5.22.

Together with the adaptive approach the number of texture updates is kept to a minimum. For every frame update, it is determined whether the values in the texture have to be updated (i.e. the dirty bit of the slicer is set):

- the slicer has been manipulated relative to the data space
- the data space has been manipulated (slicer doesn't move with the data space)
- the data values in the data space have been updated (e.g. new time step)
- the color mapper has been manipulated

If the slicer is "dirty", it is tested whether the slicer has an intersection with the data space object. In the case it has, the content of the slicer must be updated. Therefore data values at the probing points inside of the volume are calculated.

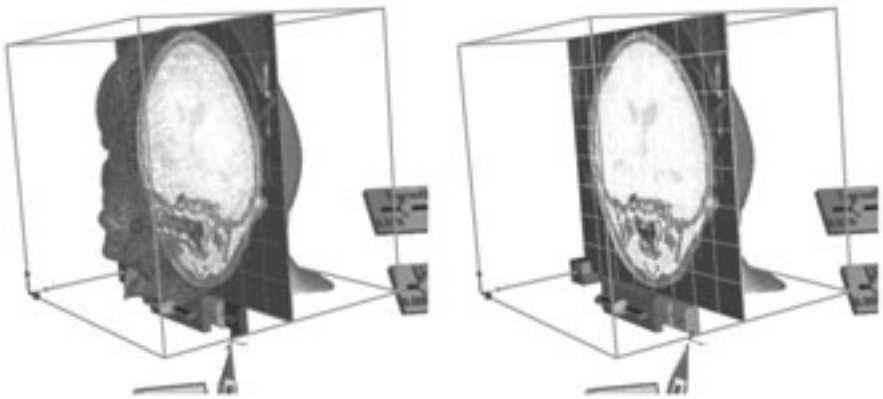


Figure 5.23: VRX - exploration of medical CT scan: iso-surface clipping tool combined with a data slicer. The wire-frame of the geometry remaining after clipping (left) can be switched off eventually (right).

First, for every probing point, the position in the data space is determined. As the slicer and the data space each use their own coordinate systems, the coordinate of each point is mapped to the coordinate system of the data space using matrix calculations. If the point is outside the data space, the texture point (texel) is set to a standard color and transparency. If the point is inside the data space, the data value of the requested dataset is determined. This value is calculated using a tri-linear interpolation in the grid of data values. Using a color mapper, which is described later in this section, the probed values are converted to color values of the textured quad.

A promising approach to data slicing is using OpenGL hardware accelerated 3D textures. A brief investigation of this approach was reported in [de Haan, 2002]. It might be used to achieve higher performance on high resolution texture slicers. However, other slicers (such as the landscape slicer and the vector slicer)

do not use texture mapping but do use the described probing and interpolation. Thus, the texture slicer, as described here, does not use the 3D textures.

Landscape Slicer

Landscape slicer is a type of slicer which uses a mesh geometry instead of a texture, see Figure 5.24. The mesh consists of an array of triangle strips, a geometric primitive that can be rendered at maximum performance. The probing algorithm is identical to that used in the Texture slicer.

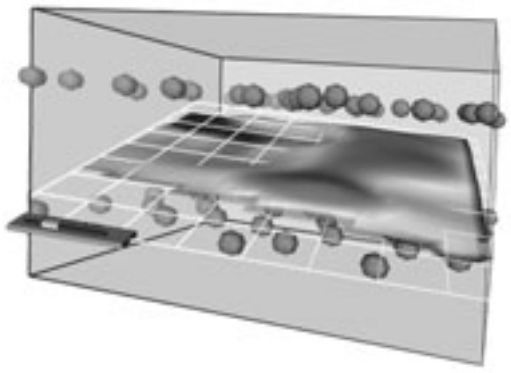


Figure 5.24: MD simulation of β-alumina electrolyte: the landscape slicer (resolution: 64x64) shows an electric field. A grid indicates the reference plane.

Data values at the vertices of the mesh are mapped into both height and color. The differences in the height of the vertices results in a "landscape-like" mesh, usually called a height field. The height values of the mesh emphasize the variations and relations in the data. Although the use of height might seem advantageous, the user should be made aware that the height is a pure virtual information enhancement. In case of the texture slicer, the color value information spatially coincides with the probed location. In case of the landscape slicer however, the visible landscape mesh cuts through points in the data space which are not the probed locations. The data values are probed in the zero height of the slicing plane and not at different heights in the landscape. We have noticed that users wrongly interpreted the visualized information. To prevent this, we have used grid lines to indicate the position and orientation of the probing plane (Figure 5.24). The rendering performance differences compared to the fast texture slicer were only noticeable when using higher resolutions (e.g. 256*256 and above) or when using many landscape slicers. An advantage of the landscape slicer, compared to the texture slicer, is that the human eye is better in detecting differences in shapes than in colors.

Vector Slicer

Another variation of the data slicer is the vector slicer (similar techniques are called arrow-plot or hedgehog), see Figure 5.25. Just as the texture slicer and the landscape slicer are used to visualize scalar data sources, the vector slicer can be used for the visualization of vector data sources.

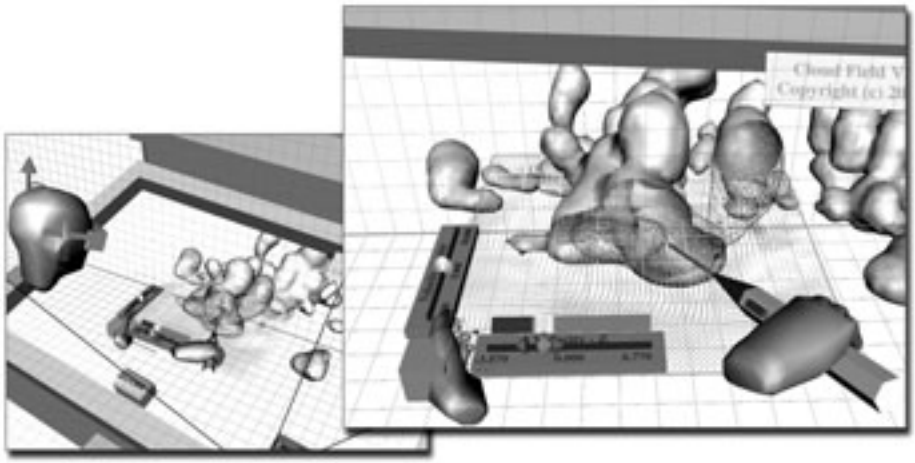


Figure 5.25: Visualization of cloud dataset: the Plexipad has attached two clipping planes and controls the vector data slicer. Velocity vectors (u, v, w) are visualized in a given slice. Vertical velocity w is mapped onto the vector's color.

Instead of using multiple polygons per object, such as pyramids or arrow shapes, we use simple thin lines (3 pixels wide) to visualize both direction and magnitude of the vector data. To indicate the difference between the start point and end point of a vector line, the end point is made semi-transparent. Because of this transparency (in combination with the multi-sampling and anti-aliasing techniques), the line has the appearance of a very thin pointed needle or arrow. The start point of each line is on a grid point on the surface of the slicer. Each probed vector is transformed from the data space coordinate system to the slicer coordinate system.

Moreover, the global scaling factor of all vectors (the length of the arrows) can be adjusted by the user. The endpoint of the line is obtained by summing the corrected vector and the start point coordinates. The result is that the displayed lines are always aligned with the direction of the vector field, and are not influenced by the orientation of the slicer. (Vector lines are placed in the SlicerDCS and not in the SystemDCS of the data space.) The scaling factor of the lines, can be interactively adjusted by using a slider widget attached to the slicer, see Figures 5.25 and 5.26.

Color mapping can be used to color the generated lines. The length of the vector can be used as an input for a color mapper in order to emphasize the vector magnitude with color. Another option is to use a different data value for the coloring of the vectors. In this way color is used to visualize another property. An example of this is the flow velocity in the case study of cumulus clouds. In this special case, researchers were interested in the direction of the flow and especially in the vertical velocity. Only the vertical component of the velocity vector was used for the color mapping. The spatial direction in the data could be determined from the arrows, while the color value emphasized whether the flow is directed upwards or downwards, see Figure 5.25.

The user can manipulate the slicer by ray-casting or directly with the stylus. Because the slicer object is very thin, the selection and manipulation using normal direct selection (positioning the stylus inside the object) would be almost impossible. For this purpose, each slicer is equipped with a control element, a box in the lower left corner. To enhance the interactive and natural use of the slicer, it can be directly controlled by the Plexipad. We like to call this a direct data slicer, see Figures 5.25 and 5.26.

The user can use the slicer to quickly scan the entire data for important phenomena. Besides this natural placement of the slicer plane, this technique enables other two-handed interaction schemes, as described in Section 5.1. In addition, related tools such as the color mapper or the previously mentioned slider widget can be attached directly to the slicer tool (Figure 5.25). These attachments allow a direct control of the behaviour of the slicer without a context switch. The user does not have to look away from the data slicer to interact with the widgets, but has the widget directly within reach.

Point Cloud

Although a data slicers can provide a detailed impression of slices through the volumetric data, users sometimes demand a complete 3D view of the volumetric data. To facilitate this, we can use direct volume visualization techniques as described below. Inspired by the use of pixels in a particle visualization test, we have tested the application of points to create a simple but fast display of volume information. This and similar techniques are called point cloud, galaxy viewer or point splatting [Swan *et al.*, 1997]. The main advantages of this technique are that its generation and updates are fast, and the density of the point cloud can have a direct relation to a physical property (e.g. density or concentration).

The main problem with this technique are the ”gaps” in the visualization, caused by the lack of connectivity between points. To achieve a ”filled” view on the data, the individual points must overlap slightly (Figure 5.26 left). This can be achieved by either enlarging the individual pixels (until the maximum pixel size of a point in OpenGL is reached), increasing the number of points

or shrinking the region to be visualized. Thus, the point placement algorithm and/or the hardware pixel fill rate will limit both the performance and use of this solution.

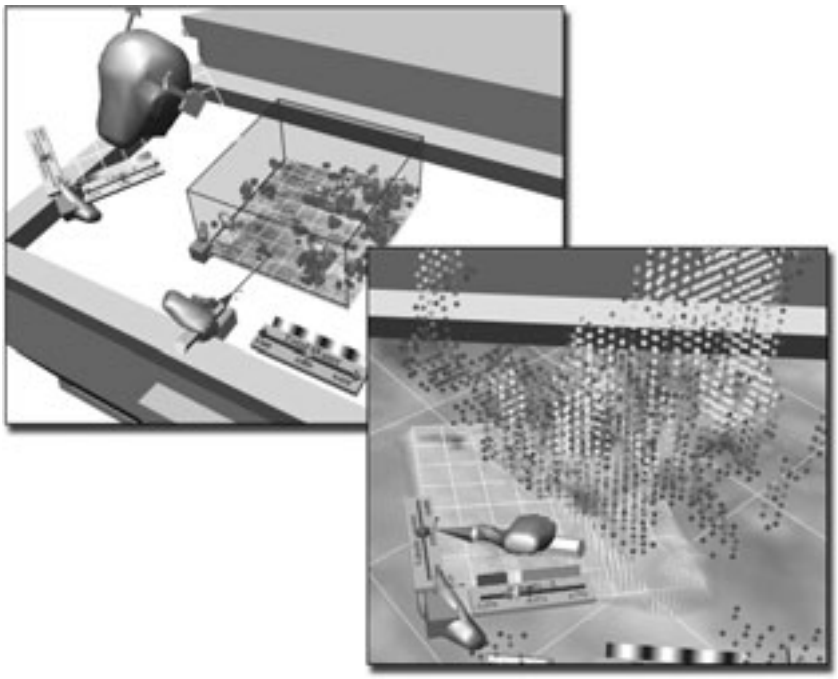


Figure 5.26: VRX - Point cloud visualization: in the left image the points overlap, resulting in an acceptable rendering quality. In the right image, the individual points are distinguishable.

Initially, we used colored and transparent points simply positioned at all grid points of the volumetric data. By enlarging the point size or shrinking the data space, the points can fill the entire space. To reduce the moiré patterns caused by the structured grid, the points can be drawn with a random offset from the original grid point (jittering). The colors of the points are determined by the color mapper based on their data value in the grid.

As an improvement, instead of placing points on every grid point of the volume data, points were only drawn if the respective data value would fall within the color mapper range (see next section). In this way, the number of points could be reduced substantially. By adjusting the color mapper, only the data of interest could be visualized. Especially when the points in the data slightly overlap and the color mapper is well calibrated, this technique provided a fast visualization of the volumetric data.

This technique would require pre-processing of the dataset in order to distribute and resize individual points in space dynamically. In this way, the most

effective positions in space are used to ensure a maximum visible quality. This process would be time-critical, taking into account the requested frame rate to calculate the maximum amount of points allowed.

Color Mapping

In most visualization tools we use color to visualize data values. We have created a color mapper class to provide an easy-to-use coloring scheme for the visualization tools. We chose to use a simple one dimensional control of the color mapping, as it is very difficult for users to work with the three dimensional control of colors in for example the HSV cone or RGB cube. During initialization of the VE, several color mapper objects can be created. For the creation of a color mapper object, the user must specify a data source on which the object works. In addition, the user indicates the desired color spectrum and mapping function. After this initialization, various visualization tools can use the provided color mapping function: given a data value, it will return an RGB color value.

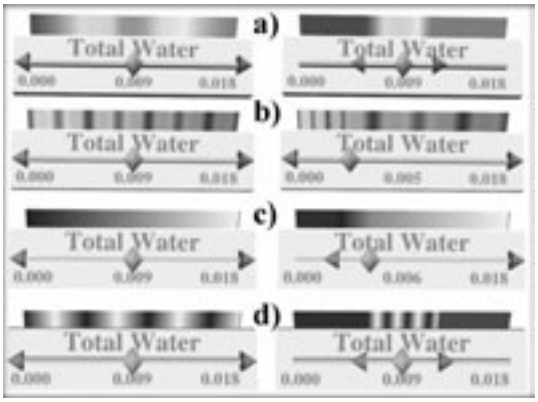


Figure 5.27: VRX color mapper widgets: the initial configurations of four different mapping functions (left) and the adjusted configurations (right); from top to bottom: rainbow spectrum (a), periodic rainbow (b), gray scale spectrum (c), periodic gray scale (d)

To interact with the color mapper object and its color range, a color mapper widget can be connected (Figure 5.27). This widget shows the color legend of the connected color mapper. It also allows the user to interact with the mapping domain of the color mapper. In our implementation, two sliders are used to define the local minimum and maximum data value limits for which a color value is returned. A third slider is used to define the distribution of the color spectrum over the data values, see Figure 5.28

During initialization, the minimum and maximum data values of the connected data source are read. These values serve as absolute limits of the color mapper. After initialization, local limits can be (re)set to specify the mapping domain in more detail. These limits are used as a reference for the input value and define the mapping domain. If a color request is issued, the relative position of the input value in the mapping domain is calculated. This relative position [0..1] is used to determine the resulting color value from the color range.

We use color mapping functions for the definition of the color range. Another approach is to use a pre-calculated color lookup table, instead of calculating the color value many times. For a given input value [0..1], the color mapping functions return a 24 bit RGB color value, 8 bits per color channel. We have defined several color mapping functions, of which the linearized rainbow function is the most frequently used. This function uses an increase in the hue of the HSV color space to generate a comprehensive color range from blue to red via green and yellow. Besides these linear functions, we have also used sinusoidal or triangular color mapping functions to create a periodic color spectrum.

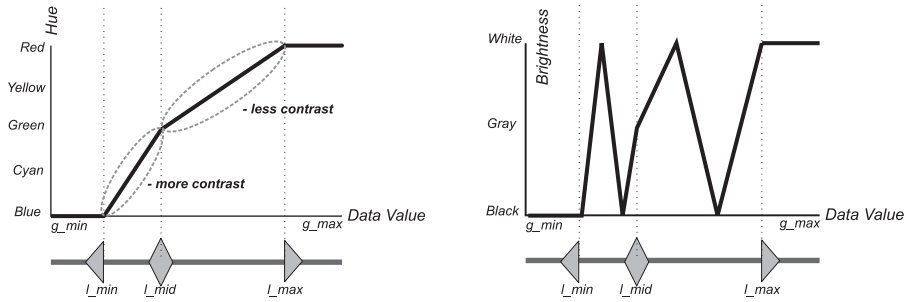


Figure 5.28: The graphs indicate the mapping (transfer) function of the rainbow spectrum (left) and the periodic gray-scale color mapper (right).

These periodic color functions can be used to reveal gradient information in the data. The use of complex color spectra can enhance the visualization in special cases. For example, the use of four different color mapping functions is shown on the cumulus cloud data in Section 6.3 in Figure 6.51, and reproduced also in the Color Section.

For each color mapper, a horizontal and vertical data slicer is used. It can be seen that in the vertical slices, the periodic color mappers reveal far more information about the gradients in the dataset, compared to the direct rainbow or gray-scale color mapping. However, disadvantage of periodic mapping is that several data values correspond with the same color.

5.2.4 Advanced Visualization Techniques

Although we have implemented some visualization tools of our own, we have also put effort into integrating existing visualization systems in our applications. The advantage of this integration is that we could use previous work of researchers and developers around the world. In some cases, a disadvantage could be a reduced performance for interactive use in VR.

Visualization Toolkit (VTK)

VTK [Schroeder *et al.*, 1999] is an open source, freely available software system for 3D computer graphics, image processing, and visualization. It supports a variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as polygon reduction, mesh smoothing, cutting, contouring, and triangulation. VTK consists of an object oriented C++ class library with support for other (interpreted) languages such as Tcl/Tk, Java, and Python.

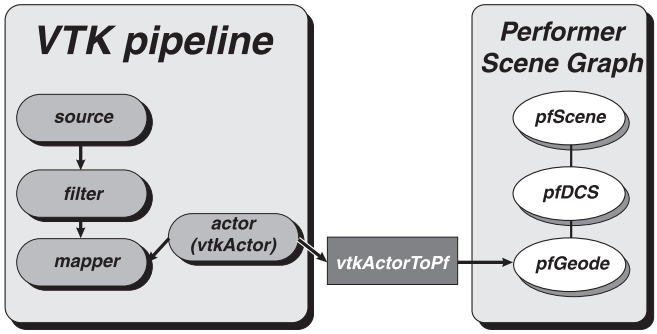


Figure 5.29: Conversion of graphical objects from VTK to Performer

The VTK uses a visualization pipeline architecture. Creating visualization applications is done by constructing a data pipeline and creating graphics objects. Data source modules are connected to filtering and data mapping modules. Many different types of sources, filters, and mappers are available, depending on the type of data and the desired functionality. Typically, the graphics objects *(vtkActors)* are connected with mappers, which deliver graphic geometries, and are rendered on the screen. Instead of using a standard OpenGL render window, we used a converter to graphics objects for our Performer applications. This converter function, *vtkActorToPf* [Web-VTK-to-Perf], converts the *vtkActors* to a Performer node *(pfGeode)* in the scene graph, see Figure 5.29.

Among other visualization techniques, we have used VTK for generation of iso-surfaces, see Figure 5.30. The data sources of the visualization application are read by the VTK pipeline. We follow the VTK data format standard to avoid time consuming conversion of data. The vertices and polygons are generated by the Marching Cubes algorithm of VTK. These surfaces can be smoothed and decimated or simplified. After processing of the VTK pipeline, the geometry and its properties are transferred to a *pfGeode* and placed directly into the scene graph under the data space object *(SystemDCS).*

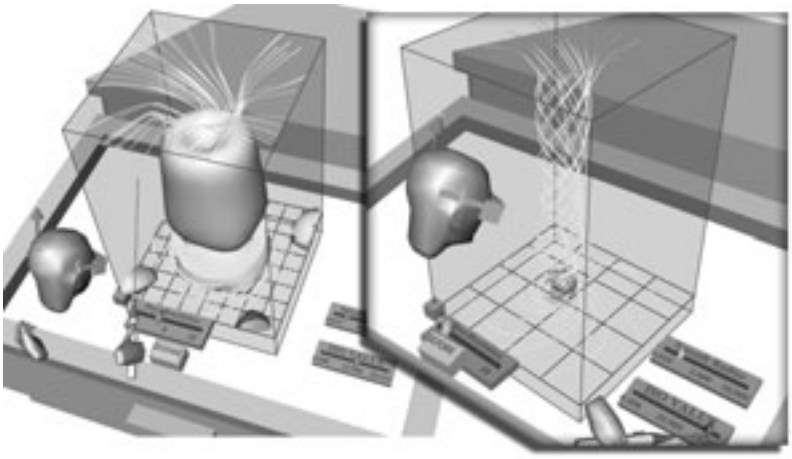


Figure 5.30: VRX visualization of the tornado dataset: iso-surfaces (vertical velocity) and streamlines starting from a plane-source; lower iso-level (left) and iso-surface of high vertical velocity (right).

Just like the other VRX visualization tools, the VTK pipeline is executed as a separate thread process in the COMPUTE stage. Although this relieves the time-critical main loop, the latency of the iso-surface generation can still be problematic. Especially when using large volumetric datasets the geometry generation may take more than 1000 milliseconds. In interactive situations, such as when the user adapts the iso-value of the iso-surface, the latency in the visual feedback hinders the execution of interactive exploration tasks. More about application of these techniques can be found in the Case Studies, Section 6.3.

A very useful visualization technique for vector data fields is the streamline tool. Using VTK we have implemented an interactive streamline tool for VR, see Figures 5.30 and 5.31.

Similar to data slicers, we use a slicer object to define a slicing plane. The streamlines are generated from a plane or line source that is attached to a data slicer object. Integration of streamlines is also rather time-consuming and therefore we use here also the adaptive resolution of the streamline sources, see Figure 5.32.

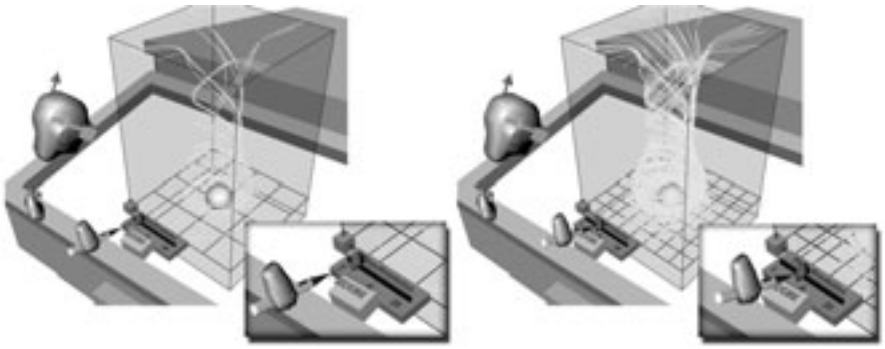


Figure 5.31: The number of streamlines generated from the plane source can be interactively chosen via the plane-resolution slider widget.

During manipulation with the streamline plane-source we use a lower resolution: 2x2 instead of 5x5. In the lower resolution we use twice as long integration time-step for calculation of the streamlines. For better usability of this technique, the plane-source of the streamlines is constrained to move only in the vertical direction.

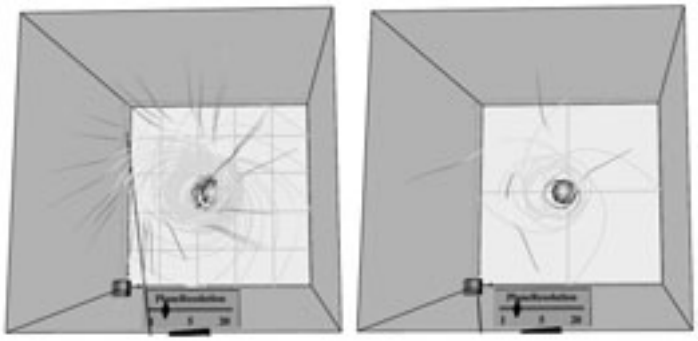


Figure 5.32: Adaptive resolution of the streamline plane-source; lower resolution during manipulation (right) and higher resolution when not moving (left)

In the same fashion as with the data slicers, the streamline tool can also be connected to the Plexipad, see Figure 5.33. Moreover, to see the interior of the clouds we have developed a specialized geometry clipping tool, which can be connected to any data slicer or to the streamline tool. We used these techniques on the cumulus cloud visualization; see Section 6.3 for more details of these techniques.

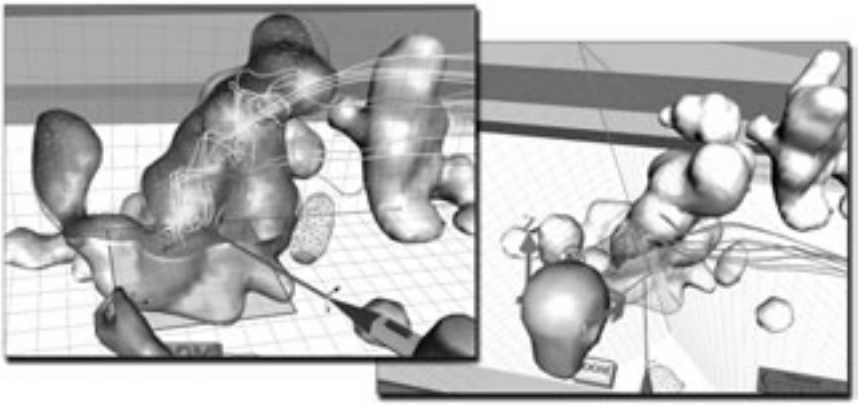


Figure 5.33: VRX visualization of the cloud dataset: a line source of streamlines is attached to the Plexipad.

5.2.5 VRX: Summary and Discussion

In the concept of the VRX toolkit the virtual environment contains the data space object, which visually represents a volume containing multi-dimensional data. This object can be freely manipulated (translated, rotated, scaled/zoomed) to get a proper view on the data. In case of a time-dependent dataset, the VE also contains time controller object for navigation through the time-steps. We have developed a toolbox for interaction and exploratory visualization. The user can take any tool from the VRX toolbox and apply it directly to the data space object in the VE on the Workbench. Also, it must be specified on which scalar or vector quantity of the multi-dimensional data the tool should work. The tool is activated when it is inserted into the data space.

The VRX toolbox contains tools such as: zoom tool, mini system tool, point probe (Click-iso-tool), line probe (LOI), ROI, plane probe (scalar, vector and landscape slicers), point-cloud tool, iso-surface tool, geometry-clipping tool, and streamline plane-source / line-source generator. The visualization tools are supported by specially designed color mappers and widgets for controlling of the parameters of the tools. The design and development of the tools was driven by user demands in our applications. Certainly, the toolbox is not yet complete, and we could easily define a long list additional tools in our toolbox. In this sense, the VRX toolkit benefits from the integration with VTK, which can offer a large number of tools needed for data visualization.

There is one problem associated with using universal visualization libraries in VR, which in practice often needs re-implementation or at least adaptation of the visualization technique for a virtual environment. Visualization in VR should be interactive. To wait several seconds for results of a given visualization

technique is too much for many applications. Visualization tools that are used in VEs should not lower the interactive response of the VE. When the generation of visualization results cannot be completed within 200 ms, the user should get a visual or acoustic feedback about the processing.

Indeed, the visualization computations must be de-coupled from the main loop of the VR application, so that fluent navigation and interaction with the VE is preserved. We solve this problem with a separate COMPUTE process, from which a threaded sub-process starts for each visualization tool. For this purpose there is only one CPU left on the 4-processor SGI Onyx2; the other three are used for APP, CULL, DRAW processes. This way the VE will keep its fluent interactive response. Only the user has to wait for completing the visualization computations and updating the geometry in the VE.

To accelerate the visualization tools we could either use parallel computing and subdivide the visualization data domain over several CPUs, or to use lower resolution of the data or the tools. Due to the limited local processing resources (only one CPU left), we did not implement the first solution in VRX. A better possibility would be to perform distributed visualization on the available HPαC supercomputers. But we leave this option for future work. For example, we see interactive and parallel generation of iso-surfaces as a great challenge.

Instead of parallelization, we have implemented adaptive resolution schemes for our visualization techniques. And this approach proved successful. The user is exploring the data space with any slicer tool, which may be attached to the Plexipad or to the data space object. When the tool is manipulated (translated, rotated) by the user, a low resolution of the tool is used. When the tool is released or not in motion/manipulation, a higher resolution is used. It usually takes a moment of time before the refined visualization is updated in the VE. This works very well with the scalar/vectors data slicers as well as with the streamline plane-source generator. For clarity, we must repeat that this technique does not down-scale the data. The resolution of the data remains intact.

A further success of visualization tools for VR also depends on the complexity of interaction and overall intuitivity of the technique. The interaction and exploration techniques, as described in Section 5.1, are indeed part of the VRX toolkit. Many of the tools were developed during the MolDRIVE project (Section 6.2) and were later incorporated in VRX.

We have validated our concept of interactive exploration on various types of data using the case studies. With VRX we could interactively visualize a large multi-dimensional time-dependent dataset in the cumulus clouds case study, see Section 6.3.

Chapter 6

Case Studies: Visualization in VR

The main goal of this research was to design a virtual environment for data visualization on the Virtual Workbench. We have developed several useful interaction, exploration and visualization tools for VR. Several M.Sc. students have also participated and contributed to this research. To demonstrate and to test the concepts of this thesis we have worked on several case studies from various application domains. The three main case studies deal with data originating from scientific simulations. Due to the 3D nature of the data we expect that their visualization and exploration in a VE on the Workbench will be more effective, leading to a better insight of the simulated process, compared to the 3D visualization on desktop workstations. As user interaction plays a key role in using the VE, we devote ample attention to it also in our applications.

One of the early applications was the flooding risk simulation, provided by WL|Delft Hydraulics. Our task was to develop an interactive 3D visualization of inundation scenarios on the Virtual Workbench. The Workbench should offer a broad overview of the flooded landscape. The user should be directly interacting with the 3D map of the country, and previewing the simulated process of flooding in case of a dike-breaking. More on this in Section 6.1. For clarity we have to mention that this application was implemented with an early version of the RWB Library, and the VRX toolkit did not exist yet.

Together with the Computational Physics (CP) group *(Faculty of Applied Sciences, TU Delft)* we have worked on development of a VR system for visualization of remotely running real-time Molecular Dynamics simulations. The contribution of the CP group was building an interface around the MD simulations. Our part of the project was to develop the visualization system and integrate the steering environment. Together we have developed the MolDRIVE system. More on this in Section 6.2. After the successful implementation of MolDRIVE on the Virtual Workbench, it has recently (summer 2002) been adapted for the CAVE.

Parallel with the MD visualization case study we have worked on the VRX visualization toolkit, see Section 5.2. Many of the interaction, visualization and exploration tools that were developed in the MD case study were later build in the VRX toolkit.

Another large project was the visualization of atmospheric data, originating from cumulus cloud simulations. In this project we cooperated with the Thermal and Fluids Sciences group *(also from Faculty of Applied Sciences, TU Delft)*. They delivered us a large multi-dimensional time-dependent dataset. From the initial desktop visualizations of this dataset it became clear that this very complex and highly turbulent data field could be more effectively studied in an interactive virtual environment than on a desktop workstation. This case study has been fully implemented with the VRX toolkit. More on this in Section 6.3.

An interesting aspect of the VR applications that we have worked on is the different scale-level of the scientific simulations. The Molecular Dynamics deals with simulation of atomic and molecular systems that are in the order of 10^{-9}m (nanometers) in size, and with simulation time-step in the order of 10^{-15}s (femto-seconds). The Large Eddy Simulation that simulates the behaviour of cumulus clouds deals with a data volume in the order of 10^{3}m (kilometers) and it uses time-steps of several seconds. Both the microscopic and the macroscopic types of simulations can be scaled into a suitable virtual environment, which is displayed on the Responsive Workbench and interactively explored by the user.

Although the "easy" scaling of virtual worlds, containing the data and their visualization, seems to be a significant advantage of the RWB concept, the users should be always aware of the scaling, especially during interaction with virtual worlds that are physically-based. For example, in particle steering of atoms in MD simulations the user can cause with "non-careful" steering movements un-physical states of the simulated system, and even cause the simulation to crash. In other words, in general the user interaction does not scale linearly with the scaling of the virtual world. Therefore users should be trained to perform certain actions properly and safely.

These rather different case studies demonstrate the Responsive Workbench concept for visualization of the scientific data. Moreover, the Workbench can provide a natural environment for steering of real-time simulations. This way the users can perform simulated "virtual laboratory experiments".

6.1 Flooding Risk Simulation and Visualization

6.1.1 Introduction to Flooding Simulations

The Netherlands has been faced for centuries with the threat of floodings. The water danger is not only coming from the North Sea but also from all the rivers that have their delta in this rather flat country. People believe that the today's costal protection, is enough to protect this country from the sea, and that the natural disaster of 1953, when large parts of Zeeland were flooded and thousands of people died, will be never repeated, see Figure 6.2(a).

Figure 6.1: Floodings in Holland, 17th century

The Dutch people have always fought with the nature to win dry land. They continue their efforts of making polders from places, where formerly there was only water. Large areas of the mainland are below the sea level. Due to such a flat and low landscape profile, the rivers need more time to reach the sea and naturally need more space around the river-basins. Sometimes even the water in the lower parts does not flow in the 'right' direction.

(a) (b)

Figure 6.2: Recent large-scale floodings in the Netherlands: Zeeland 1953(a), Limburg 1995(b)

The management of costal waters and rivers is a very complex discipline and for the Netherlands it is of vital importance. Although complex systems of dikes and inundation zones around the rivers were built, the potential danger of flooding at some places of this country is still very high. Such places are mostly around the big rivers like the Rhine, the Maas and the IJssel. In the province of Limburg quite regularly the river Maas floods her banks, see Figure 6.2(b).

There is a great need for flooding risk simulations. We have cooperated with WL|Delft Hydraulics [Web-DelftHydraulics]. Among other activities they also conduct research on the flooding and inland water systems. They also develop systems for hydrodynamic simulation and flood early warning systems.

Within this case study we were asked to develop an interactive 3D visualization of the flooding simulation for the Responsive Workbench. This VR system should give a broad overview of the land with pre-computed flooding scenarios. The user should be able to interactively explore the effects of the flooding and analyze the flooding risk at various locations.

6.1.2 2D Visualization of Flooding

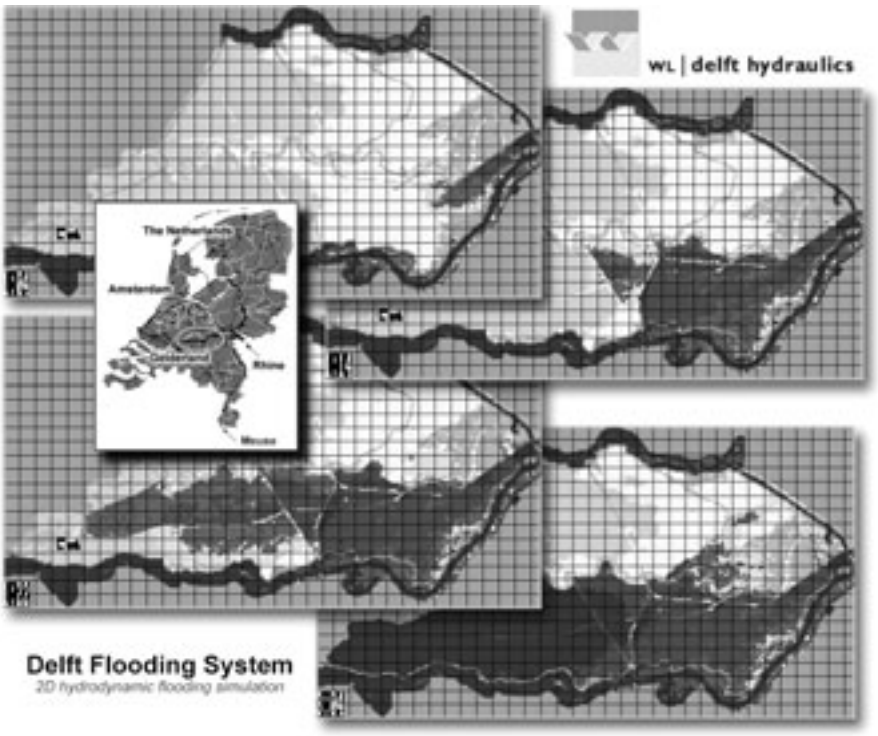


Figure 6.3: 2D hydrodynamic flooding simulation [WL| Delft Hydraulics]

We were provided with the data of DFLS simulation (Delft Flooding System). We were also introduced to 2D visualization that was used at WL|Delft Hydraulics, see Figure 6.3. In this case study we have visualized flooding scenarios in the province of Gelderland. The area of this flooding simulation is about 11x17 km, including towns such as Gorinchem and Tiel. It is surrounded by a complex dike system. The rivers and canals around that form a potential danger are: the Waal, the Lek/Nederrijn, and the Amsterdam-Rijnkanaal. Several scenarios of dike-breaking were simulated on a 2D grid with dimensions 211x318. One grid cell has real-world dimensions of 50 meters.

The hydrodynamic simulation is based on a 3D elevation model of the landscape, the water saturation in the soil, water levels and water flow in the rivers. As an initial condition a break-point in a dike must be specified. The flooding simulation generates new water levels at grid points per time-step. At the beginning the flooding data file contains the initial water level at each grid point. For compression of this data file, only changes of water levels per time-step are stored into data file, see the following fragment of the data:

```
4321   0   0  // time-step number, "0 0" identifies a new time-step
 187 20   8  // x, y, z-increment of water level
 264198   7  // all 3-digit integers (except x or time-step: 4 digits)
 267200   7
4322   0   0
 262193   7
 267200   8
4323   0   0
 187 20   7
 256181   7
 138197   7
```

In early times of this project we were provided with the water level files in ASCII with 9 different water levels (0..8). In the original 2D visualization the height of the water was visualized with 8 different colors. Although it seemed enough for 2D visualization, at an early stage of the project we have very encountered problems with inconsistency of the absolute water levels and the absolute height of the terrain. The water geometries simply did not fit on the 3D landscape. At that time the water levels were given relative to the height of the terrain.

Therefore, we requested and obtained binary files with floating-point levels in centimeters and defined in absolute coordinates. Then both geometries finally matched with each other.

6.1.3 Prototype of 3D Visualization

Together with the water level data file(s) we received a 3D elevation field, from which the landscape could be reconstructed. We have created a prototype 3D

visualization in AVS 5.0. Several new AVS visualization modules were implemented for generation of the 3D landscape from the height field data and for visualization and animation of the flooding data (water levels), see Figure 6.4.

Figure 6.4: Visualization network of the prototype 3D visualization of flooding data in AVS 5.0

We have used the standard AVS module *field-to-mesh* to generate the 3D geometry of the water levels and to generate the 3D landscape from the height field, see Figure 6.5. Using a simple animation scheme, we could preview the process of flooding in 3D. Unfortunately, the incremental compression scheme for the water level data allowed only sequential reading of the data.

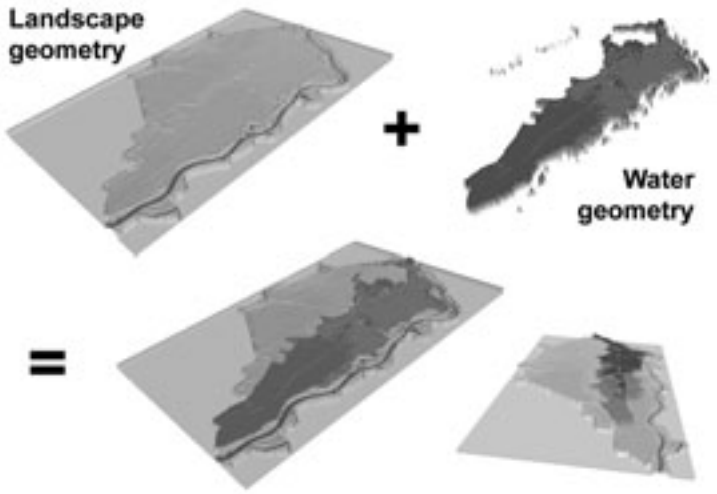


Figure 6.5: Prototype 3D visualization of flooding data in AVS 5.0

We have implemented random access to any time-step of the flooding simulation, but it was rather slow because a switch to another time-step means to re-read the whole water level file from the beginning to the chosen time-step. For the VR version of the flooding visualization we have used therefore separated data files for each time-step, which was much more suitable for a random access.

6.1.4 3D Visualization in VR

From the prototype visualization we have learned that online generation and visualization of complex landscape and water geometries is not a good approach to VR visualization. Generation of the full 3D mesh takes too much time and cannot be performed interactively from the VE. One solution to this is to preprocess the whole flooding simulation and generate geometries for each time-step.

Landscape Geometry Visualization

Another problem of visualization in VR is the complexity of the geometry that has to be rendered. The available OpenGL hardware of the Onyx2 (Infinite Reality 2) can render practically about 5 millions of triangles per second, which leaves us about 100-150 thousand triangles in the whole scene for maintaining screen refresh rate of 2x25Hz.

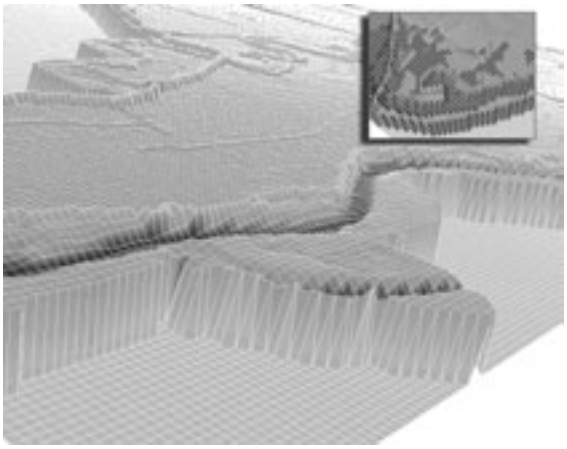


Figure 6.6: Original mesh (211x338) of the 3D elevation model

The mesh 211x318 has 142.636 triangles (Figure 6.6), which saturates the rendering capacity. Moreover, when we would like to render the water levels also as a 3D mesh then the drop in the refresh rate (<<10Hz) would cause distortion of the immersion in the virtual environment.

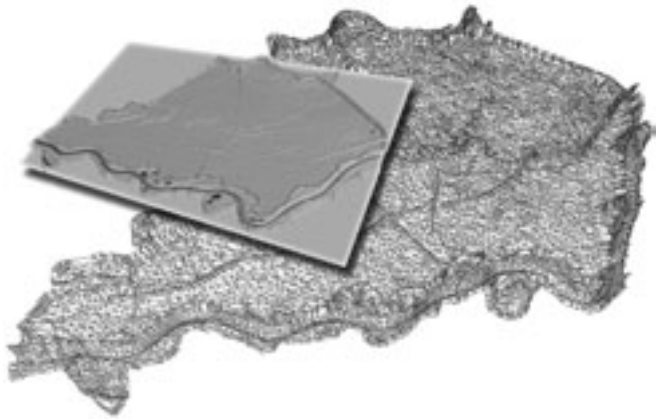
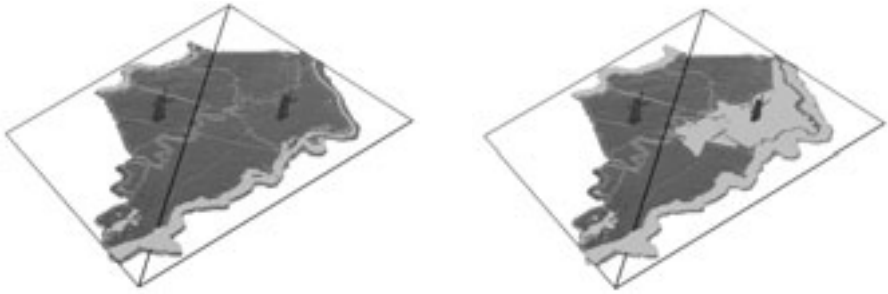


Figure 6.7: Decimated landscape model was colorized with a high-resolution texture.

In our approach we have simplified the complex 3D mesh using Qslim [Garland, 1999] to 30.000 triangles. Due to this simplification fine details of the landscape disappeared. To partially restore the detailed features of the landscape we have used the full-resolution texture (211x338 resized for OpenGL into 512x512) for the decimated landscape geometry, see Figure 6.7. The rendering frame-rate is much faster than 50Hz and it leaves rendering resources for other graphical objects.

Water-level Geometry Visualization

The differences of a few decimeters in the height of the water are practically invisible when observing the landscape of 11x17km on the Virtual Workbench. The water surface can be seen as a plane and it can be approximated with a textured rectangle (quad) instead of a complex geometry.



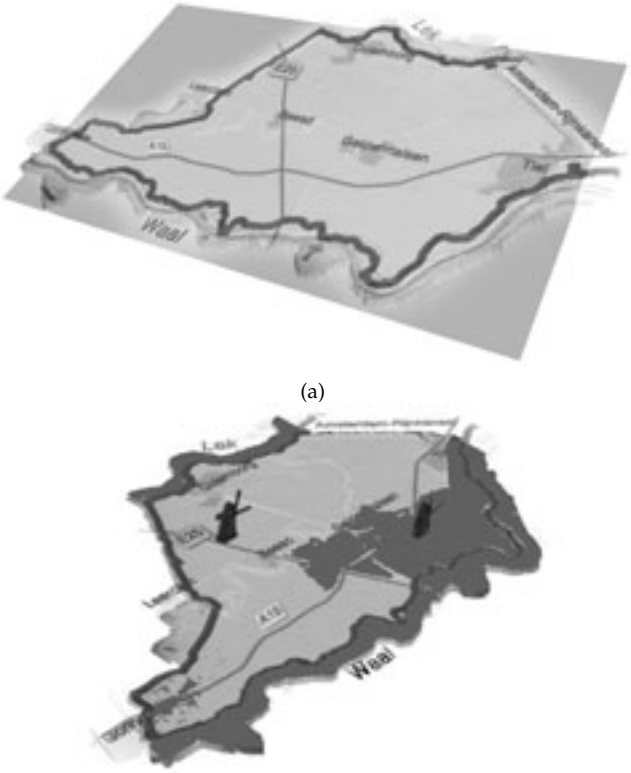
(a) (b)

Figure 6.8: Water-levels are visualized with a textured rectangle: fully-saturated rivers (a), the dike is broken and the land is being flooded (b).

The fine differences in water levels can be much more efficiently visualized by color in the texture. The whole flooding scenario has been pre-processed and texture files generated for each time-step.

Visualization Layers

Additional textured layers can be used for topographical and geographical information, see Figure 6.9. This way we can emphasize important and strategic infrastructure that are important for flooding risk analysis, like cities, roads, highways, railroads, dikes, power plants, etc.



(a)

(b)

Figure 6.9: Visualization layers: decimated and textured landscape geometry with a topographical layer (a) and the water layer (b).

Interactive Visualization on the Responsive Workbench

On the Responsive Workbench the user has a good overview of the area that is being studied for the flooding risk. Several scenarios with different places, where the dike could break, were simulated and explored in the virtual environment. The user can choose one of the simulated scenarios and interactively play back the flooding, see Figure 6.10. In the worst-case scenario the whole area (11x17 km) can be inundated in 4 days.

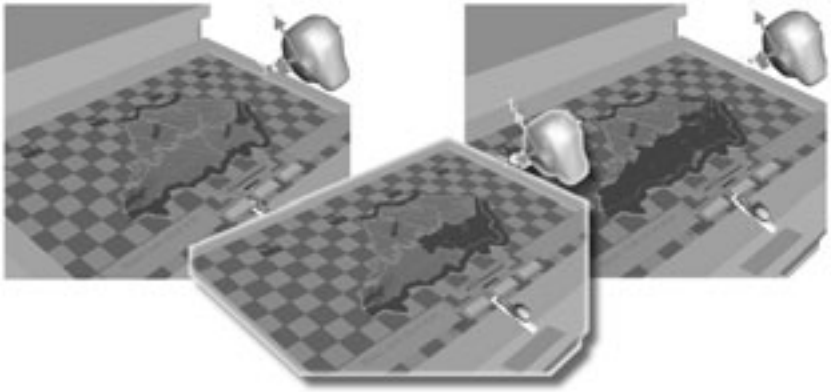


Figure 6.10: Interactive visualization on the Responsive Workbench: the user can play back in 3D the flooding simulation.

6.1.5 Flooding Visualization: Summary and Discussion

We worked on the flooding visualization in the first year of this PhD research when we were intensively developing the RWB Library & Simulator. Although the results of this case study are not as important as the results of the other two case studies, we mention it mainly as our introductory VR application. At that time we didn't have the VRX toolkit yet and we searched inspiration in visualization packages such as AVS 5.0, AVS Express, and VTK. In fact, we developed a stand-alone visualization system for the flooding data on the RWB. We have faced traditional problems of VR such as: generating of the VR content, reduction of the complexity of the scene (polygon decimation and using textures), and user interaction in the VE. This case study showed that the Responsive Workbench can be efficiently employed for visualization of simulations that can be projected on top of the geographical data and the landscape geometry, such as the flooding risk simulations.

As future work we should mention the interactive selection of the dike breaking points and other flooding conditions, but this needs real-time flooding simulation or at least that the simulation delivers the data within a few minutes. Another improvement could be a hierarchical representation of the landscape that would contain roads and cities and would facilitate local detailed views on the flooded areas.

6.2 Interactive Visualization of Molecular Dynamics

This section describes in detail the MolDRIVE visualization and steering system for Molecular Dynamics in VR and its applications. We have already given a brief overview of this system in Section 4.3, mainly describing the simulation steering aspects and the particle steering tools.

We have worked together with several M.Sc. students on this case study; it was presented in two M.Sc. theses [van Hees & den Hertog, 2002; de Haan, 2002]. Van Hees and den Hertog worked mainly on the interface between the MD simulations and the Workbench VE. De Haan worked on the visualization and interaction techniques, and the design of the VRX visualization environment on the Virtual Workbench.

In the next section we will give a brief introduction to Molecular Dynamics, describing our motivation for using VR in this project.

6.2.1 Introduction to Molecular Dynamics

Molecular Dynamics (MD) is a technique used to simulate the properties of complex materials and molecular structures. MD helps to gain insight in the material properties and the inter-molecular interactions. This includes finding and understanding spatial positioning and interactions of atoms and molecules, but also properties of materials such as molecular structure, conduction and diffusion. To be able to describe or predict these properties we try to understand the processes involved.

The physical behaviour of individual atoms in a material can typically be described by Newton's equations of motion. These equations express the behavior of atoms in time, based on their position and momentum, and the influence of atomic forces, caused by the interaction with other atoms. The purpose of any MD simulation program is to numerically solve the equations for all individual particles in the computational system. The particle properties such as position and momentum are obtained by numerical integrations of these equations using discrete time steps. To be able to correctly sample atomic motions, the length of the time step is chosen sufficiently small, mostly in the order of 10^{-15}s (femtosecond). To observe simulations with long time characteristics (in the order of 10^{-9}s), many time steps are required.

The initial problem of MD is to create an atomic/molecular model of the material. The atoms or molecules are represented as individual objects or particles, each with its position in space. The initial MD model configuration can be constructed from physical measurements such as NMR (Nuclear Magnetic Resonance) or by using MD modeling programs.

The constructed model is used as the initial configuration in the MD simulation program. Usually, the simulations run for thousands of time steps while intermediate results are written to hard disk in so-called *trajectory files*.

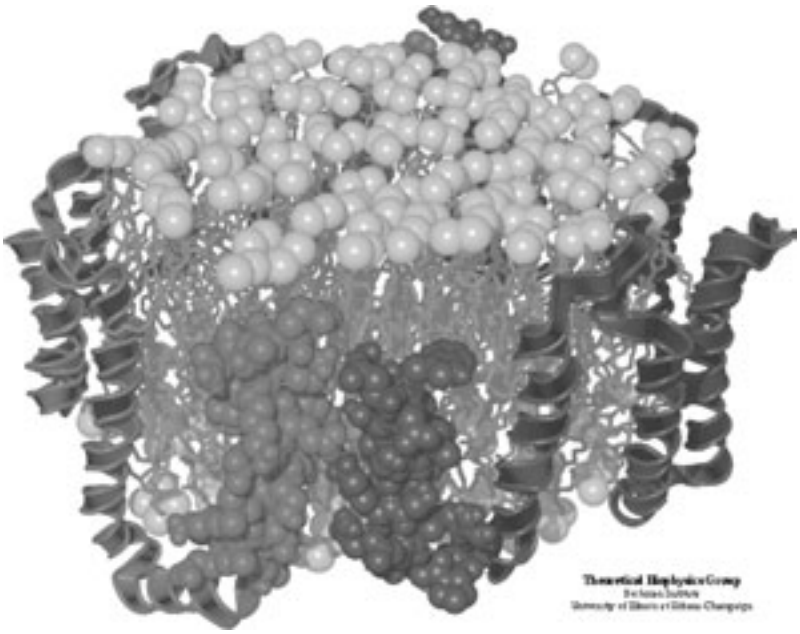


Figure 6.11: Molecular Dynamics example [Image source: Web-VMD]

Typically, MD simulations can take in the order of hours to several months or even years, depending on the complexity of the model, the available computational power and naturally the number of time steps. An example of a simulated molecular structure is shown in Figure 6.11.

To get an overview of the behaviour of the particles or a specific material property in the simulation, these trajectory files are usually analyzed afterwards (off-line). The interaction with the simulation is mostly done in advance of the simulation run by adjusting input parameters such as the initial atomic configuration or the temperature. The influence of these parameters on the simulation is analyzed after the simulation is completed. Interesting behaviour and global parameters such as diffusion and particle flow can be extracted from these files and visualized in a regular graph.

In addition, the atomic configuration in each of the static trajectory files can be visualized using existing software packages and techniques for the visualization of atomic and molecular data (e.g. VMD [Humphrey *et al.*, 1996], RASMOL [Sayle & Milner-White, 1995]).

In Figure 6.12, several basic graphical MD representations are shown. Each of the representations provides a different level-of-detail view on the (complex) molecular structure. By using suitable visualization methods, scientists can study their theories and predictions on material properties, using the simulation for verification.

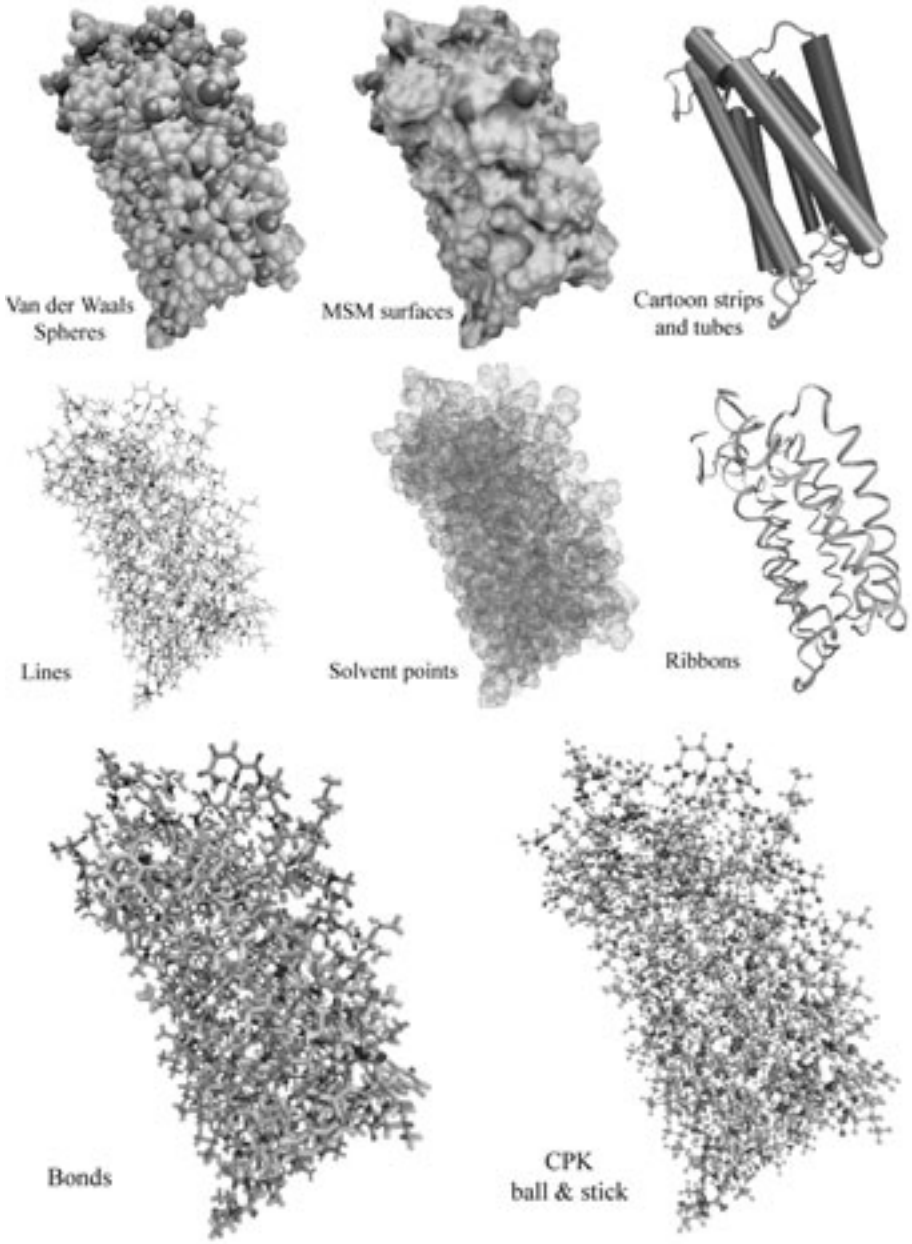


Figure 6.12: Basic graphical MD representations, showing Bacteriorhodopsin biomolecules with 3,700 atoms [Image source: Web-VMD]

The MD researchers need to simulate larger molecular structures, and to analyze slow processes on larger time scales. The use of larger, more realistic atomic models of a material in simulation also allows more reliable and accurate measurements of the material's properties. For these reasons, the goal of an MD application is to simulate as many particles as possible as fast as possible, which requires a significant amount of computational power. Several approaches are used to speed up the simulations.

One approach is to reduce the amount of complex calculations by adapting the actual algorithm, which involves physical simplification while a certain accuracy is attained. An example is the PPPM method [Beckers, 1999], where the more complex force calculations are performed only on particles that are within a certain distance *(Particle-Particle interaction within cut-off range),* while the influence from more distant particles is calculated from a mesh *(Particle-Mesh interaction on a long range).*

The software approach is to optimize the compilation and execution of the computer program. The hardware approach is to simply increase the speed and performance of the CPUs that perform the numerical calculation.

Another trend is to use distributed computing, dividing the computational tasks over several processors or computers (simulation nodes) and allowing parallel execution of the simulation program. Today's powerful MD programs (such as i.e. NAMD [Web-MD-namd], Gromacs [Web-MD-gromacs], DEMMPSI [Web-MD-demmpsi], etc.) support distributed and parallel computing.

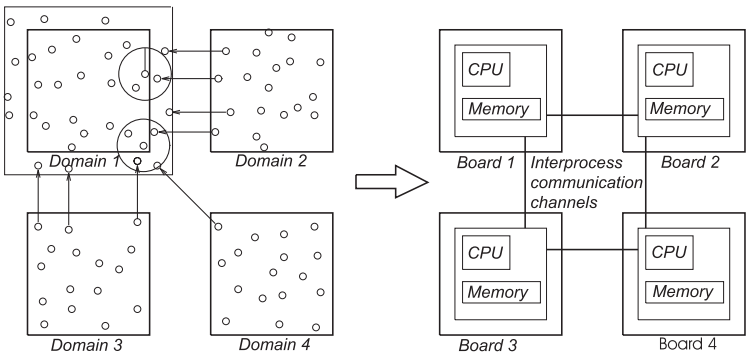


Figure 6.13: The (spatial) domain decomposition of MD simulation

Two decomposition methods are used with MD simulations. *Particle decomposition* assigns unique particles to processors and the particles remain with the assigned processor throughout the whole simulation.

Figure 6.13 shows a schematic overview of the *domain decomposition.* In this technique the spatial domain is divided over the available simulation nodes. Each of the nodes simulates behaviour of the particles that fall within the domain. For a proper simulation of the particle behaviour at the domain boundaries it is necessary to simulate interactions between particles in the neighboring domains. Also it is important to support moving particles from one domain (one CPU) to other domain (other CPU).

The development of efficient numerical algorithms and realistic physical models have made Molecular Dynamics a well established research tool, which is now widely used in (bio-) chemistry, solid state physics, materials science and many other areas.

In this case study we want to demonstrate that the Virtual Workbench can offer a better visualization environment with a more effective interaction interface for displaying and analyzing of the complex MD structures compared to the conventional desktop systems.

6.2.2 Introduction to the MolDRIVE Project

The growing interest in steering capabilities in MD simulations has resulted in the development of several systems such as Steered Molecular Dynamics (SMD) [Leech *et al.*, 1996], and Interactive Molecular Dynamics (IMD) [Stone *et al.*, 2001]. These systems are based on NAMD [Web-MD-namd] simulation software and VMD [Humphrey *et al.*, 1996] visualization software, and offer both (desktop and semi-VR) visualization and steering of atoms and molecules. We decided not to use these software packages for several reasons. First, our colleagues from the CP group were not using NAMD software. They have concluded [Beckers, 1999] that for the problems, they were studying, DEMMPSI and Gromacs were better. We did some experiments with VMD and we decided to build a more intuitive visualization system, which fully exploits the advantages of the VEs.

The results of the simulation parameters on the behaviour of the particles are commonly analyzed after the simulation has been completed. The off-line interaction with the simulation is useful for the analysis of long-term behaviour and global properties of the simulated phenomenon. Current trends in MD studies show a growing importance in the interactive steering capabilities on a small time-scale, for example in protein design, in molecular docking, and in the study of energy configurations. In this project we want to enable this interactive study of particle behaviour and material properties. This requires that we can visualize and steer running real-time simulations.

An Early Approach: Vrsim

In 2001, Vrsim [Garvic, 2001], a prototype VR visualization of real-time MD simulations on the Virtual Workbench, was developed at the Computational Physics group at TU Delft. Although this solution has some design flaws, the results of controlling a real-time simulation in a VE on the RWB were promising. Our goal was to design a new approach, exploiting the advantages of VR and MD techniques to the fullest extent.

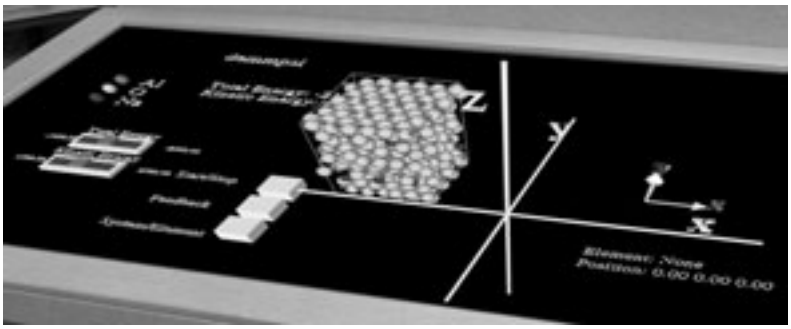


Figure 6.14: Vrsim on the Virtual Workbench

Vrsim allowed users to view a running MD simulation in a virtual environment on the Responsive Workbench, see Figure 6.14. A simple approach to steer the simulation was also implemented.

The Vrsim application consists of three software components: the simulation component, the visualization component and the databroker, see Figure 6.15.

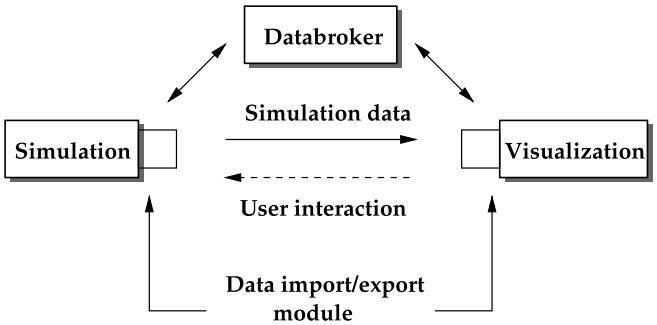


Figure 6.15: The Vrsim system overview

The whole system, including the simulation, was designed to run on an SGI Onyx2. The simulation component used DEMMPSI MD package [Web-MD-demmpsi]. The databroker controlled the communication and the data-flow between the simulation and the visualization. The visualization component provided an interactive view of the simulation data on the Virtual Workbench.

Vrsim allowed real-time visualization of MD simulations on the RWB of particle systems up to 700 particles. Larger particle systems result in a slow scene update rate of the virtual environment, severely disturbing the VR experience of the user. The DEMMPSI simulation was marked as the main bottleneck of the system, slowing down the entire application. Vrsim was not implemented using RWB Library and Simulator, had very simple VE with limited user interaction, and the simulation module with the databroker were constantly causing overload of the visualization server. Further, Vrsim was not very stable. Among other decisive arguments of this project the following requirements became our motivation for the design of a new system:

Improvement of simulation and communication: the development of a distributed visualization and steering environment; the simulation should run on parallel supercomputers and communicate with the visualization computer via a network.

Improvement of visualization: new and improved visualization and interaction techniques should be developed to provide the user with a more effective visualization and steering environment.

More detailed analysis of Vrsim and motivation to implement a new system can be found in [van Hees & den Hertog, 2002; de Haan, 2002].

Used MD Simulation Programs and Available Supercomputers

In the MolDRIVE project we have chosen to work with the following MD simulation packages:

- **DEMMPSI:** MD software package [Web-MD-demmpsi], internally developed at the Computational Physics group, with actual research applications on solid state electrolyte materials. Furthermore, it is reasonably fast and allows the use of parallel computing using MPI (Message Passing Interface) libraries.
- **Gromacs:** MD software package [Web-MD-gromacs], primarily designed for biochemical molecules like proteins, lipids and DNA. This software is also very fast, widely used and the code is freely available and well documented. For parallel computing, traditionally PVM (Parallel Virtual Machine) is used. Optionally, MPI can be also implemented within the Gromacs code.

In this project we used some of the supercomputers (Figure 6.16) available at the High Performance Applied Computing Center (HPαC) at TU Delft:

- **The Cray T3E** consists of 128 RISC processors DEC Alpha 21164, each with 128 MB of RAM. The processors are interconnected by a fast network.
- **The Beowulf-class Linux cluster** is constructed of 27 system boards SGI 1100/1200, each hosting two Intel Pentium III processors 700 MHz, and 256 MB of RAM. The boards communicate via a regular 100Mb Ethernet.

- **The SGI Origin 2000** contains 8 MIPS processors R10000 (195 MHz). These processors have access to 768 MB of shared memory.

Figure 6.16: Supercomputers at HPαC. From left to right: Cray T3E, Beowulf Linux cluster and SGI Origin 2000

6.2.3 MolDRIVE Design Requirements

Our main goal was to design a framework, which enables users to interactively work with real-time Molecular Dynamics simulations in a virtual environment on the Responsive Workbench. We concentrate on working with large and complex MD simulations. To interactively work in a virtual environment with this large amount of real-time generated data, both the visualization component and the simulation component should have a sufficiently high update frequency.

The simulation component should calculate a simulation time step as fast as possible for a given simulation. A faster simulation enables users to study slower movements and behaviour of molecular structures on a larger timescale. Also the implications of the user's steering actions on the simulation process can be studied more effectively.

The visualization component, which generates the VE should maintain a sufficiently high scene update frequency (10 Hz minimum) to maintain the VR experience. Calculations in the visualization component and the complexity of the graphical scene have an effect on the frame rate. The goal was to create a visualization component that could generate effective visual representations of the (large) simulation data while maintaining a sufficiently high frame rate.

Independence of the frame update of the visualization component and the simulation component should be attained. Instead of waiting for simulation data of the next time step to arrive, the visualization component should continue to update the graphical scene of the VE with the highest frame rate possible, using the latest available data. In this scenario the visualization component could continue to update the VE (including the update of the head position and interaction) while the simulation is calculating a next time step concurrently on the remote supercomputers.

With the introduction of a remote simulation the performance of the entire system will not only depend on the performance limitations of the visualization component and the simulation component, but also on the communication throughput over the network. The fast supercomputers could prevent the simulation component from becoming the main performance bottleneck of the system. Instead, the visualization component or the data communication between the graphics server and the supercomputers could become a new bottleneck. In the following sections this aspect and performance measurements will be discussed in more detail.

When performing large MD simulations with many particles, we cannot simply transport and visualize the particle data. First, the screen would probably be filled with particles, from which the user cannot easily get the desired information. Second, the number of particles can become too high to be rendered fast enough by the graphical hardware. Instead of using the particle data for visualization, we should use alternative ways for the transmission and visualization of large numbers of particles. The main requirements of the MolDRIVE system can be summarized as follows:

- **Distributed simulation and visualization:** the MD simulation program running on any of the supercomputers should be extended with a communication module to be able to communicate with the visualization component through a network connection.
- **Generation of derived particle data:** concepts and algorithms to create derived/abstract data from the (large) simulation data should be investigated.
- **Visualization and interaction in VR:** the visualization component should be implemented using the RWB Library and Performer, providing an effective user interface in the VE. Users at the Responsive Workbench should be able to interact with the simulation and the particles or some other abstract particle representation like particle clouds in the case of large numbers of particles. Both interaction and visualization techniques for particles and derived data should be implemented.

The next section will describe the MolDRIVE architecture and its components.

6.2.4 Architecture and components of MolDRIVE

Based on the initial requirements we designed and implemented a new distributed system. Our system was named MolDRIVE, Molecular Dynamics in Real-time Interactive Virtual Environments. An overview of the MolDRIVE architecture is shown in Figure 6.17.

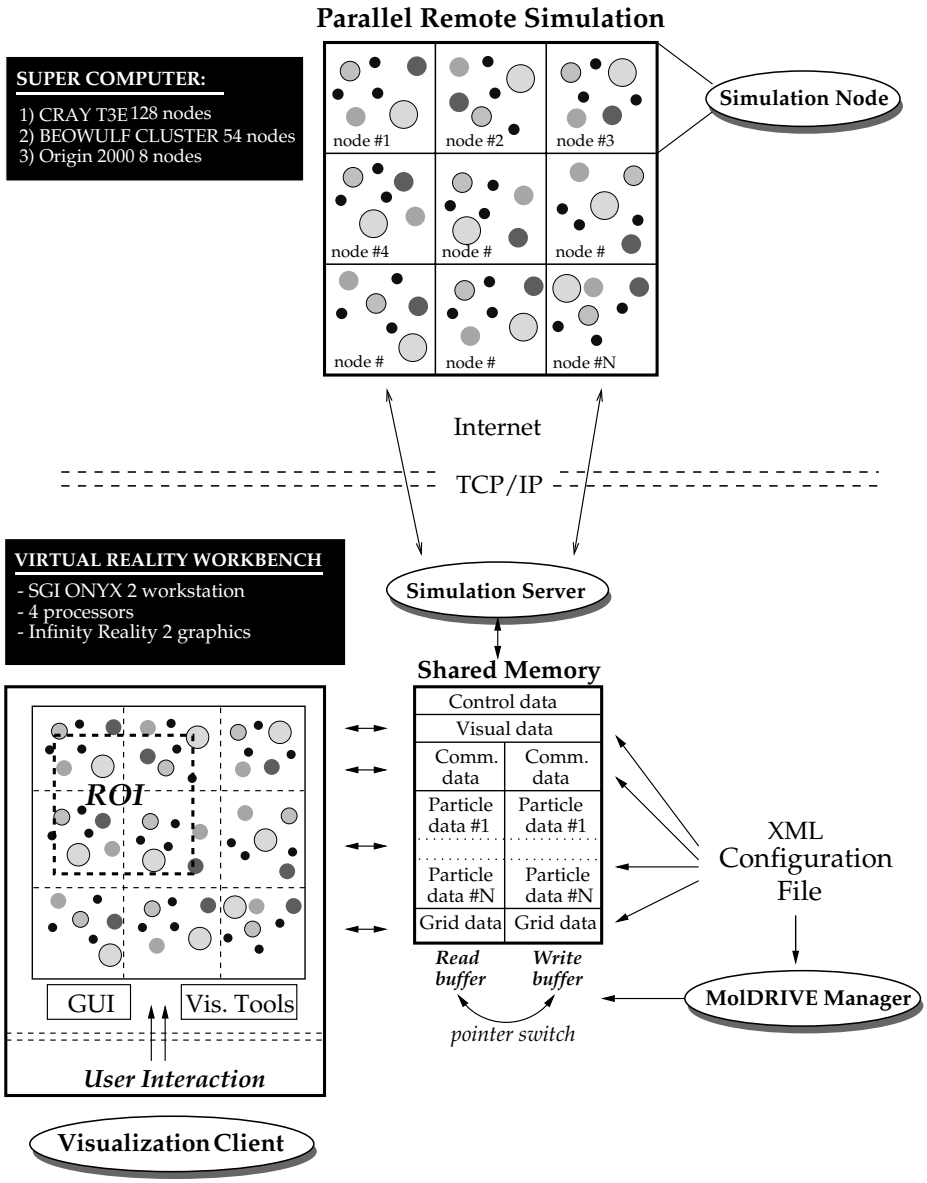


Figure 6.17: MolDRIVE architecture and system components. The MD simulation is running remotely in parallel on the supercomputers, while the visualization and the computational steering is done on the RWB.

It shows the four components of the system: *the remote Simulation Nodes, the Simulation Server, the MolDRIVE Manager, and the Visualization Client.* A simulation runs on multiple nodes of a remote parallel computer. Each of the Simulation Nodes communicates with the Simulation Server on the RWB through a TCP/IP connection. The Simulation Server provides the simulation nodes with simulation parameters, data requests and user interaction feedback. The simulation nodes return the requested data. The system is first initialized by the MolDRIVE Manager with a configuration file. Then it creates and controls appropriate Shared Memory structures through which the Simulation Server and the Visualization Client can communicate.

Double buffering enables the Simulation Server to write the simulation data it receives to one buffer while the Visualization Client simultaneously reads the simulation data from the previous time step from the other buffer. The VC controls the visualization of the simulation and handles the user interaction.

Communication between Components

Since MolDRIVE consists of four separate components, one of which is running remotely, it is important to have effective communication. The Simulation Nodes must ”know” which type of information/data is requested by the Visualization Client. For this purpose the information buffer was constructed. It uses a simple message format which defines all information about the current data request or response. If a user would like to display the force vectors of all particles in the simulation, the VC can set the value of the ”FORCES” element in the information buffer to ”1”. As the data request containing the information buffer is received by the simulation nodes, they will append the force and velocity data of all particles to the data response. The transfer of data between the components is done either by using Shared Memory (local communication) or using a network (remote communication).

Simulation Server and Simulation Nodes

The remote simulation on the parallel computer communicates with the Simulation Server component on the Onyx2 using sockets of the TCP/IP network protocol. The Simulation Server creates a separate socket endpoint for each Simulation Node and waits for incoming connections from the Simulation Nodes. After the simulation is started and initialized on the remote computer(s), each Simulation Node connects to the Simulation Server, using the configured IP address or hostname and port number on the visualization server. After the connection is established, the Simulation Nodes transmit initialization data needed by the Visualization Client, such as identification of each individual particle type.

Communication between the Simulation Nodes and the Simulation Server is performed by passing of information and data messages. At each time step the Simulation Server sends a data request in the form of an information buffer

filled by the Visualization Client. Each Simulation Node receives the information buffer, gathers the requested data from the simulation program and processes a possible user feedback. Each node replies to the Simulation Server with the updated information buffer, containing properties such as the simulation status and size of the data message it will send. Afterwards, each Simulation Node will send its data response to the Simulation Server, see Figure 6.18. The Simulation Server then places the received data in the Shared Memory for use by the Visualization Client.

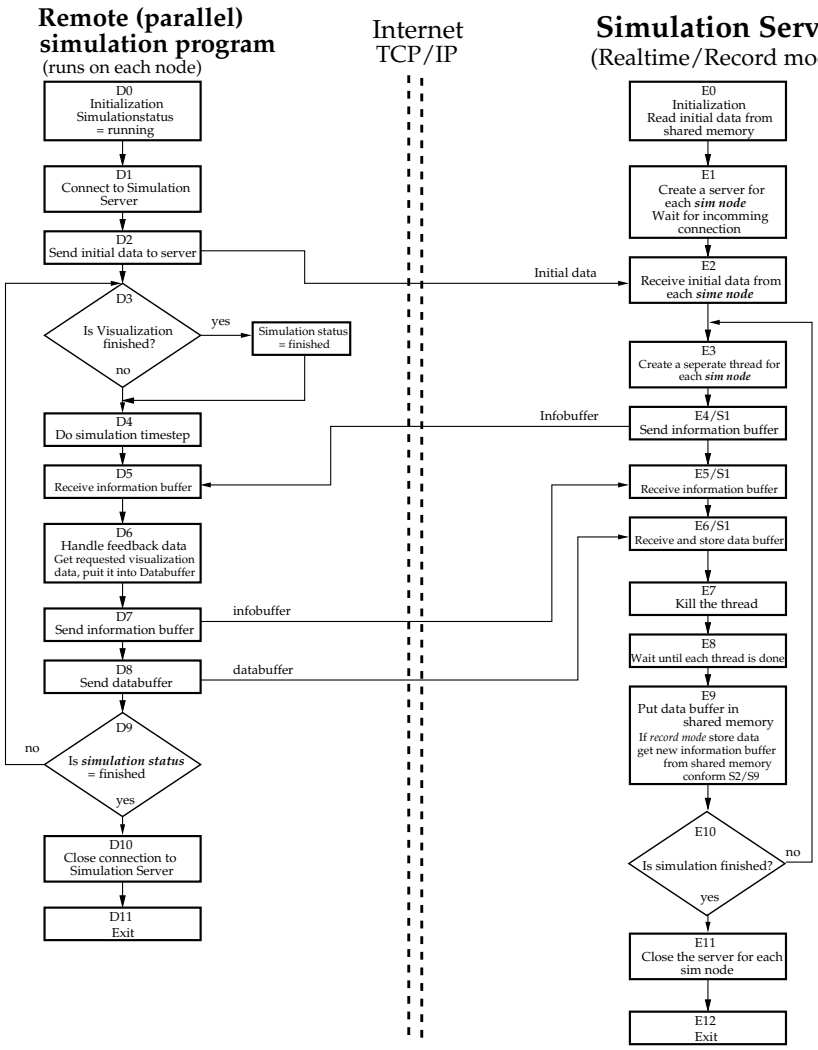


Figure 6.18: Communication structure between the Simulation Nodes (on the supercomputer) and the Simulation Server (on the visualization server).

The communication interface of the Simulation Nodes sends only the data that is requested through the information buffer to save the communication and processing time. It has access to the internal data structures and data generation functions of the MD simulation program on that node. We created functions that generate the derived spatial data (scalar or vector). Each node stores its domain data in a regular Cartesian grid, and all grids are merged in the Simulation Server. Examples are particle density, long range forces (vectors), and kinetic energy. Moreover, a Region-of-Interest (ROI) can be defined in the Visualization Client. Each node will only send particle data if the particle is located within this ROI. The communication between the nodes and the server is done simultaneously by using multi-threading, somewhat reducing the influence of network latency, see Figure 6.18. By using a thread for each communication channel with a Simulation Node, the communication with all Simulation Nodes is performed in parallel. Furthermore, the amount of transferred data is reduced by type conversion of the simulation data (8 byte float into 2 byte integer) that is sent for visualization. For visualization purposes, the simulation data notation does not have to be as accurate as for simulation purposes.

Local Communication

The communication between the MolDRIVE Manager, the Simulation Server and the Visualization Client takes place through *the Shared Memory.* All components have access to the information in this Shared Memory, see Figure 6.19. We constructed a special class to control the Shared Memory using *semaphores*, allowing a component to lock and unlock access to the data structures. This prevents data inconsistencies and errors, which are introduced when two components are simultaneously reading from and writing to the same memory address. Three buffers are created, consisting of the following elements: *control data, visualization data, dynamic simulation data.*

A small information buffer called the *control data* contains the status of all components. This buffer contains the name of the component that is reading from or writing to which buffer in the Shared Memory. To obtain read or write access on a certain buffer, the requesting component first has to gain access to the control data by trying to lock its access. If this locking succeeds, no other processes can access to this part of memory, and the process can safely read or alter data. After gathering the control information and updating on which part of the Shared Memory the component will work, the control data can be unlocked. The *visualization data* buffer contains static visualization information, such as the particle colors and dimensions. The third buffer, the *dynamic simulation data*, is used for the transmission of simulation data from the Simulation Server to the Visualization Client. Further, it is used for sending user feedback in the opposite direction. Using a *double buffering* technique on this data structure, both the Simulation Server and the Visualization Client always have their own buffer available. While the Visualization Client is reading new simulation

data in one buffer, the Simulation Server can already start filling the other buffer with new data. When both have finished their actions, the pointers to buffers are switched by the MolDRIVE Manager.

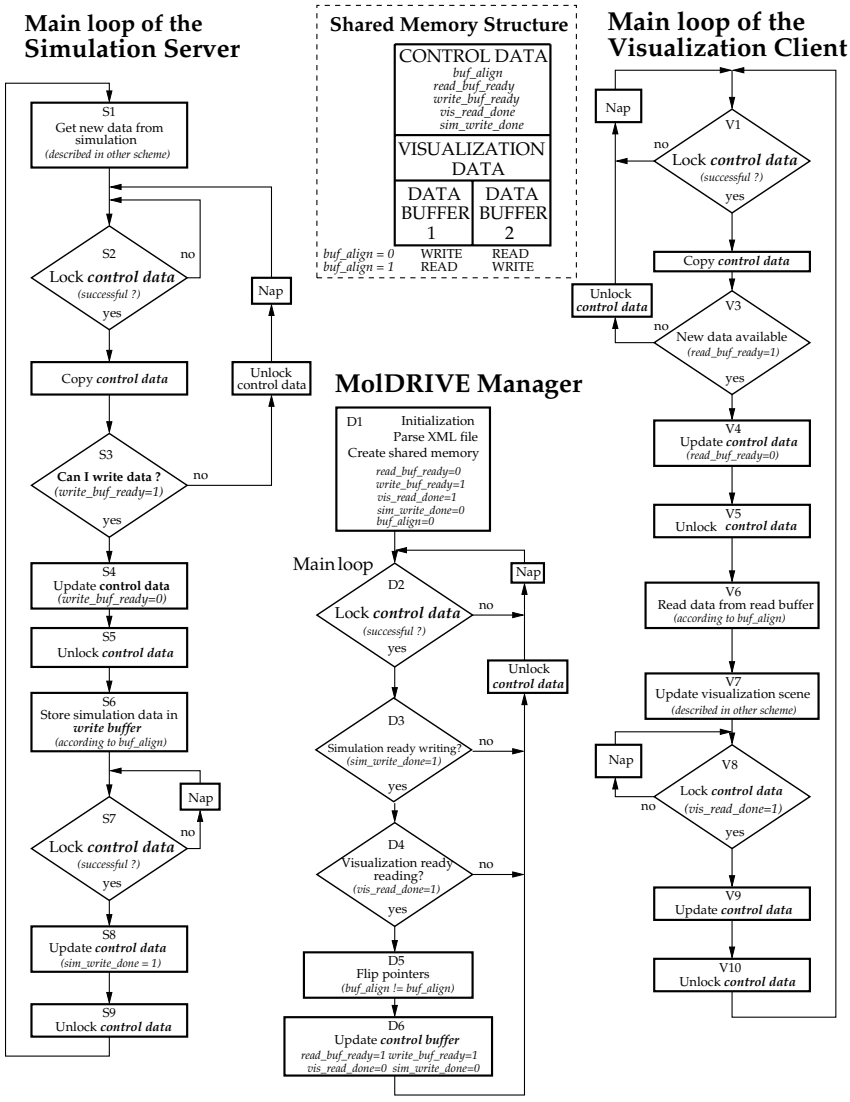


Figure 6.19: Communication structure between the MolDRIVE Manager, the Simulation Server, and the Visualization Client. All these components are running on the SGI Onyx2, using the Shared Memory.

The data buffer contains three types of data:

- **Communication data**, which contains the information buffer used for communication between the Simulation Server and the Simulation Nodes.
- **Particle data**, which contains type of atom and its dimensions, position, velocity, and force vectors.
- **Derived grid data**, which contains derived simulation data stored as 3D grids, such as density and potential fields.

From the communication data, the exact content of the entire data buffer can be retrieved. For example, from the communication data the exact dimensions of the grid data can be determined.

MolDRIVE Manager

The MolDRIVE Manager is responsible for the initialization and control of the local system components. In the initialization process a configuration file containing information on the simulation and visualization session is parsed. The configuration files are created using XML syntax (eXtensible Markup Language). More about how to write the configuration files for MolDRIVE can be found in [van Hees & den Hertog, 2002; de Haan, 2002].

According to the configuration, the required shared memory areas are allocated and the communication with other MolDRIVE components is initiated. During execution, the manager controls the swapping of the double data buffers and at the end of the MolDRIVE session cleans up the shared memory used. If one of the other components crashes, the allocated shared memory areas can still be released safely by the MolDRIVE Manager.

6.2.5 Simulation Steering and Time Control

The MolDRIVE system is not only capable of handling real-time simulation, but it also supports recording and playing back of simulation sessions. In this way, large and slow simulations can be pre-recorded and visualized afterwards in playback mode on the RWB. Although this eliminates the bottleneck of the slow simulation, there can be no direct interaction (particle steering) with the simulation. If the desired derived data of the simulation is recorded, it can be visualized using the regular visualization tools. A more advanced concept we introduced is the *instant replay*, a mix between real-time operation and playback. In this mode, the user can visualize and interact with a running (remote) simulation, while the last few hundreds of time steps are recorded simultaneously on the harddisk. When the user sees (or has just missed) an interesting phenomenon, the simulation can be paused and the recorded time steps can be played back from hard disk. During this instant replay, the user can also use exploration tools to inspect the phenomenon detected or the effect of steering actions, when particle steering was applied.

The Simulation Server is responsible for playing back and recording the simulation data files. The user controls the flow of simulation time steps by using the time control widget, see Figure 6.20.

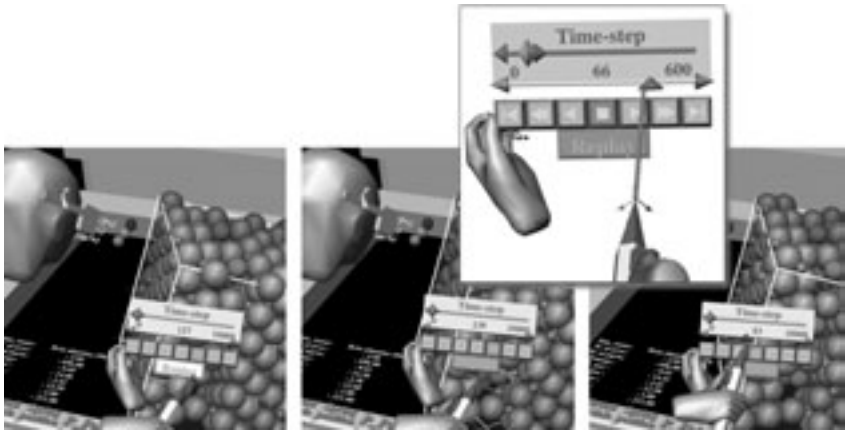


Figure 6.20: Control widget allows direct steering of the real-time simulation. Using instant-replay the real-time simulation stops and interactive playback mode is activated.

6.2.6 The Visualization Client of MolDRIVE

The Visualization Client (VC) of MolDRIVE provides a virtual environment for visualization and control of the MD simulation. As previously described, the VE of this application has been the test-bed for the development of the VRX tools. Several visualization and interaction tools used in the VC have already been described in the previous two chapters. This section will highlight the integration aspects of the MolDRIVE Visualization Client and the tools used.

Initialization

During the initialization the VC tries to connect to the MolDRIVE Manager through Shared Memory. When the connection is made, the VC will read the session-specific information e.g. how many particles and of which type, which grids for derived data, etc. Further, the information of the required visualization tools and GUI widgets is read. Based on this, the VC will create required particle objects, visualization tools and widgets, and set up the the MolDRIVE virtual environment.

Main Loop

The RWB Library and Performer control the basic visualization, manipulation and interaction aspects of the VE on the RWB. The main-loop code of the VC is implemented in the user code (see Figure 6.19), which is executed every frame by the RWB Library, and interaction code has been programmed using the RWB-Interactors, see Section 3.4.

In this main loop, the control buffers in the Shared Memory are checked for the availability of new simulation data. When the new data is ready, the VC shortly locks the control buffer and updates the local references to the data buffers to be visualized, see Figure 6.19. Simultaneously, the new data request and the user feedback is put back to the buffers. This data, for example steering or ROI data, will be used by the simulation for the generation of the new time steps.

The information in the current buffers is visualized in the VE. The VC is not completely integrated with the VRX concept, where the updates of visualization tools occur in a parallel COMPUTE process. Therefore, the update of all visualization tools are handled in this main loop. A full integration of VRX into MolDRIVE would speed up the whole system.

Virtual Environment

A typical layout of the VE of the MolDRIVE Visualization Client is shown in Figure 6.21. The main object of the VE is the data space which contains the simulation data, both the particles and the derived grid data. In addition, several data slicers can be used to visualize available grid data.

The workspace also contains several widgets, such as the time controller widget on the Plexipad, and the interaction buttons which are used to activate various RWB-Interactors. These RWB-Interactors include Particle steering tools, Region of Interest tool and the Zoom tool. The user can navigate using the Mini-system. Any of the data slicers can be attached to the Plexipad.

Visualization of Molecular Data

As already mentioned, we are dealing with two types of molecular data:

- **Particle data**: position of atoms (particles), type and dimension of atoms, force and velocity vectors of each atom (particle).
- **Derived grid data**: volumetric data that are computed on a 3D grid by the Simulation Node or by the Simulation Server; for example particle densities, kinetic and potential energies, or force fields.

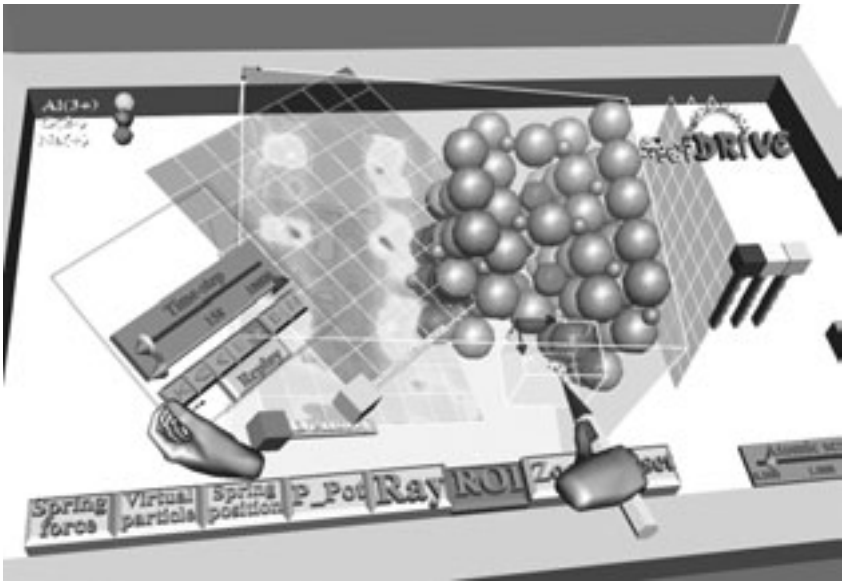


Figure 6.21: The MolDRIVE Visualization Client (see also the Color Section)

For particles we have used a spherical representation (Van der Waals' radius), see Figure 6.12. Color and radius directly correspond with the type of atom. This can be specified in the XML configuration file for each particle type. For more details, read Appendix B of [van Hees & den Hertog, 2002].

At each time step, the simulation delivers new particle data and also derived data. Which derived data are sent to the visualization server can be selected via the XML configuration file, or directly from the VE itself. For visualization of the 3D data we use direct data slicers (for scalar and vector data). When the simulation is not in real-time mode, the user can use slower visualization techniques like an iso-surface generation. As our simulation-visualization scheme has a blocking mechanism, the simulation cannot continue until the visualization has finished the visualization of the current time-step. For better throughput we use double buffering.

To visualize both the velocity and the force of all particles, we use arrows on the particles. According to the orientation and magnitude of the vector, the arrow is rotated and resized. Another feature is the *particle path* technique. This technique simply draws the trajectory of a particle in time. At every time step, a thin colored line is drawn from the previous position of the particle to its current position, using color to represent its current velocity magnitude. After some time steps, the colored trajectory of the particle is revealed. In the case study of an electrolyte material, this function is used to analyze the motion of specific ions. Figure 6.22 shows an example of the electrolyte material, where both particle vectors and particle paths are used.

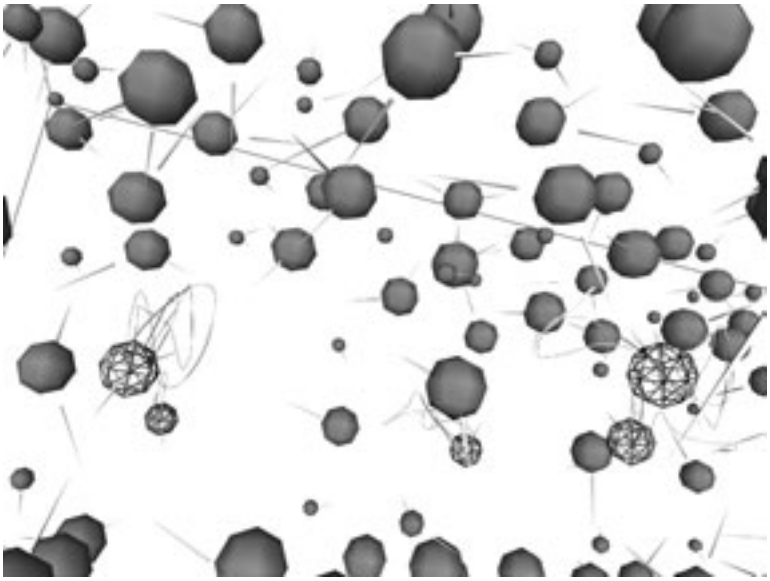


Figure 6.22: Electrolyte visualization (β-alumina). Darker arrows indicate force, white arrows indicate velocity. Particle paths are used to visualize the motion of sodium ions (wireframe).

In large particle systems we cannot see more information by simply showing all particles and their properties. Depending on the visualization task and the application area, derived volumetric data or extracted features can probably provide more important information than the use of large numbers of particles. Instead of spending too much time on optimizing the particle visualization, we have concentrated more on the development of the visualization techniques for the derived data.

6.2.7 MolDRIVE Case Studies

Electrolyte Study

In the Computational Physics group, the DEMMPSI simulation package has been used extensively to study the properties and phenomena of solid electrolytes [Beckers, 1999]. One of the main case studies in their research has been the β-alumina. We have used this material as the main case study for MolDRIVE as well. The structure of this material is characterized by the conduction planes of sodium ions and bridging oxygen atoms between dense blocks of aluminum oxide, see Figure 6.23. The ionic conduction properties material has been of interest for use in solid state batteries.

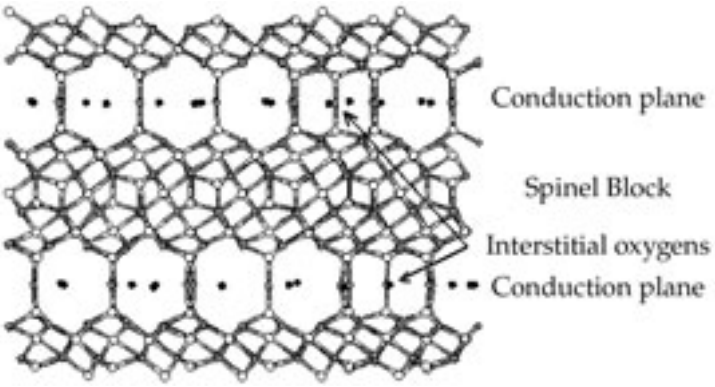
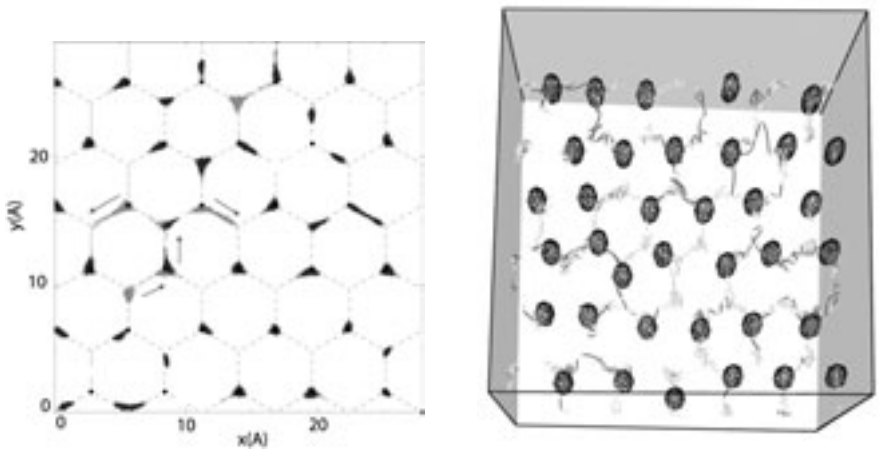


Figure 6.23: The schematic structure of β-alumina

To study the movement of the sodium ions in the conduction layer, we have used the particle path function to visualize the trajectories of these particles in time. After a short time of simulation, the movement of these ions revealed a hexagonal (or honeycomb) pattern, as was well known from the literature (Figure 6.24).

This pattern is caused by the chain reaction of moving sodium ions, which push each other away from the available equilibrium position with low potential energy. Using particle steering this chain reaction can be studied in detail, by gently pushing a sodium ion towards an occupied site and thus causing a movement of the occupying ion. The hexagonal energy pattern can also be seen during particle steering, using the assistance of a data slicer to display the potential energy of a moving particle, see Section 4.3.



(a) (b)

Figure 6.24: The characteristic movements of sodium atoms in the conduction layer. A schematic 2D projection (a) and an observation in MolDRIVE (b).

Another phenomenon was observed by the use of a direct data slicer to display the long-range forces in a simulation of β-alumina. During the visualization, a system wide oscillation pattern in the long-range forces was detected (Figure 6.25). In the MD literature, this oscillation was described as being caused by a breathing motion of the spinal blocks, and can be seen in the vibrational density of states (VDOS) spectra.

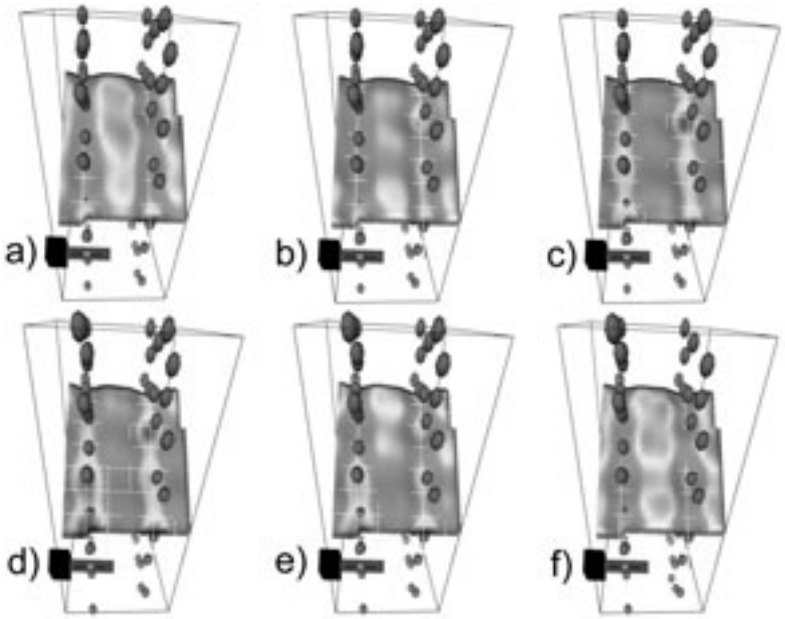
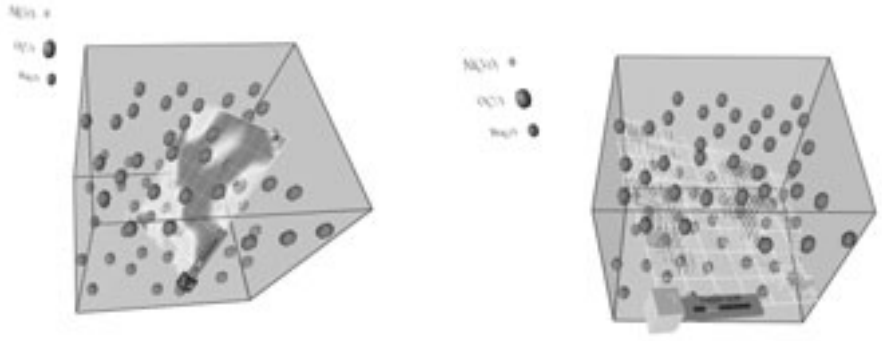


Figure 6.25: Sequence of 6 consecutive simulation time steps , where the landscape data slicer visualizes the magnitude of long-range particle forces (PPPM). This way the oscillation pattern is made visible.



(a) (b)

Figure 6.26: β-alumina electrolyte; Na^+ atoms visible only; direct data slicer shows potential field (a); vector data slicer shows PPPM force field (b) ;

Simulation and Steering of Proteins

To prove the versatility of our MolDRIVE system, we decided to integrate it with another MD simulation package. The Gromacs MD simulation package [Web-MD-gromacs] was chosen for its performance, the availability of code and the user experience in the CP group.

Initially, a file converter was constructed to convert Gromacs particle trajectory files to MolDRIVE input files. This allowed off-line visualization of original Gromacs simulations in MolDRIVE. As an example, we have converted trajectory files from a simulation of protein folding. This simulation data, the result of more than 14 months of computation with Gromacs on the SGI Origin 2000, was provided by Jaap Flohil and Loek Bakker from the Computational Physics group. Using the MolDRIVE system, the user can study the folding process in the VE in detail. By using the time control widget, the user can browse through the simulated time steps and replay interesting movements.

The MolDRIVE communication protocol was also integrated in Gromacs. This enabled real-time visualization of a remotely running Gromacs simulation. In addition, the integration allowed interactive atomic steering using our particle steering techniques. This feature is especially interesting for use in molecular modeling tasks, such as correcting, docking and (un-)folding of molecular structures.

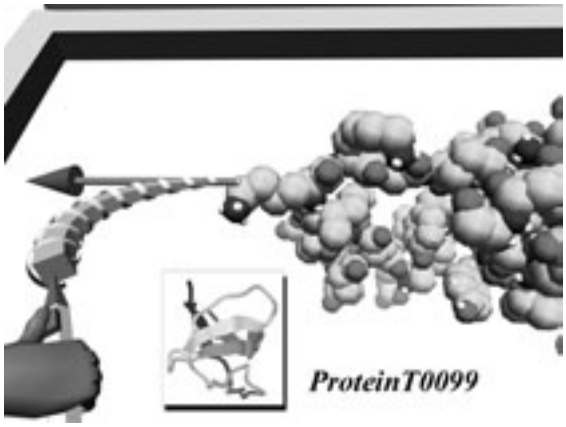


Figure 6.27: T0099-protein simulation using Gromacs; the user can unfold the protein using the spring manipulator.

The particle steering technique was used for the forced unfolding of a small protein, see Figure 6.27. This protein (T0099) contains 56 amino acids and was the smallest available protein. In this example the protein was forced to unfold by pulling the N- and C-terminal ends apart. The pulling procedure was started by the selection of a the C-atom from the backbone (the C-terminal end).

Next, a pull force was exerted on this atom by retracting the spring manipulator connected to this atom. Figure 6.27 shows that the molecule is gradually unwrapped by pulling the C terminal away from the system's center of mass.

The Visualization Client was not adapted for the integration with Gromacs. It was merely used to visualize the basic molecular structure only, using the standard sphere representation to represent the atoms in the proteins. As described earlier, other packages for molecular visualization offer several other visual representations for protein structures, see Figure 6.12. For example, the cartoon representation uses strips and tubes to indicate specific structures inside the protein (Figure 6.27). However, the particle representation was found most suitable for use with the particle steering. Moreover, the use of derived data was not implemented. The original code used in DEMMPSI for the generation of derived data requires considerable rewriting. This is because of the differences in code structure and internal data representation between DEMMPSI and Gromacs.

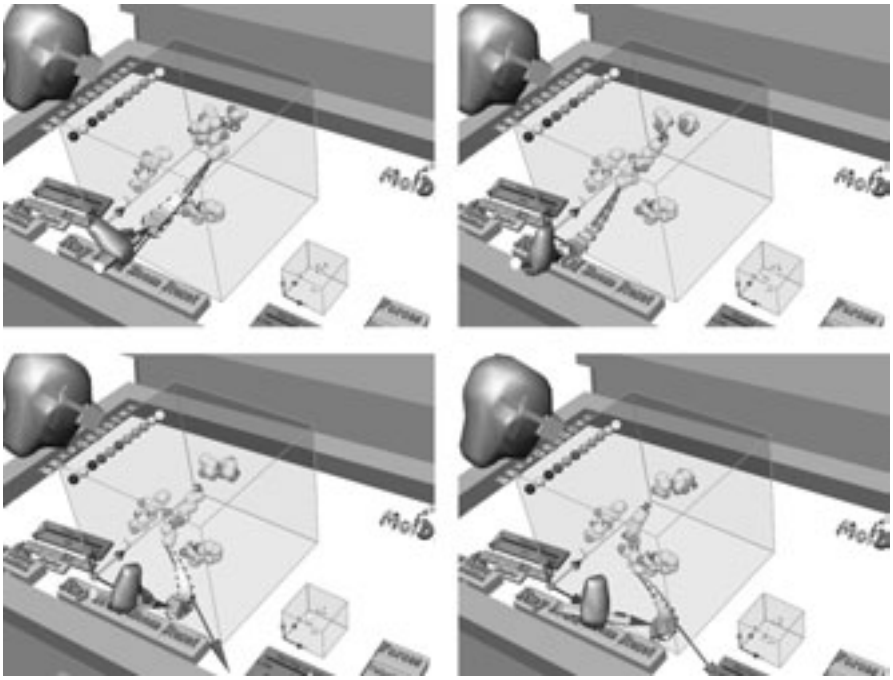


Figure 6.28: MolDRIVE (Gromacs): particle steering of protein fragment (see also the Color Section)

The promising results in our examples have led to the continuation of the use and development of the combination of Gromacs and MolDRIVE, see Figure 6.28. An extension of the interactive steering features could be of great importance for the steering of entire molecules.

MolDRIVE in the CAVE

MolDRIVE was designed originally for the laboratory table metaphor of the Virtual Workbench. After a successful presentation of the particle steering tools we were asked to give a demonstration of MolDRIVE in the CAVE. It gave us a possibility to test another type of interaction environment. MolDRIVE was implemented on the basis of the RWB Library, OpenGL and Performer. The applications in the CAVE at SARA in Amsterdam are usually based on the CAVE Library [Web-CAVElib], pure OpenGL or Performer version. The CAVE Library provides only basic functions for accessing the tracker sensors and the rendering pipeline. The CAVE Simulator enables the user to implement CAVE applications on desktop systems. The CAVE Library doesn't support any scene graph maintenance. Graphic and scene graph libraries must be used in addition to the CAVE Library.

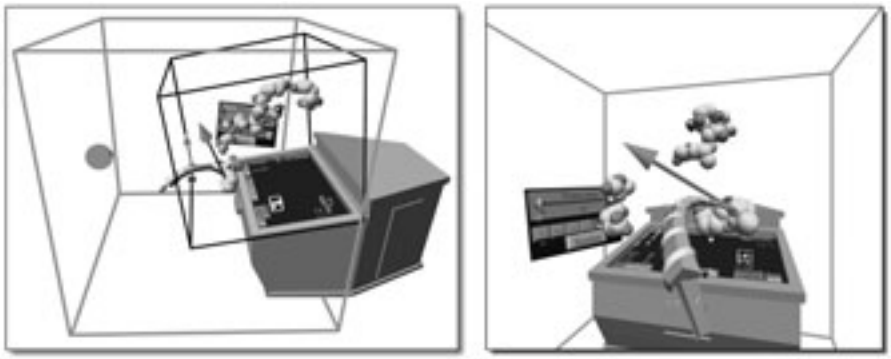


Figure 6.29: MolDRIVE in the CAVE Simulator (MD simulation - Gromacs)

To facilitate a simple conversion of MolDRIVE from the Workbench to the CAVE we have implemented in the RWB Library an additional interface for the CAVE Library (Performer version). Instead of using the tracker daemon, the tracker data are delivered by the CAVE Library. Another adjustment is that the RWB Library does not set up the Performer rendering pipeline. This is left to the CAVE Library and the CAVE projection configuration. The CAVE Performer-Library provides the main application loop for the RWB Library. These adaptations have resulted in a special version of the RWB Library for the CAVE.

An advantage of this solution is that original RWB applications can be compiled for the use in the CAVE without any change in their source code. In other words, the MolDRIVE source code remained intact. Most of the useful features of the RWB Library also work in the CAVE. As in the CAVE at SARA a Plexipad is not available, we had to place the Plexipad object freely in space. To enable easier interaction with it we have enlarged it four times.

Figure 6.30: MolDRIVE in the CAVE at SARA, Amsterdam

The CAVE experience has learned us that the surround screen projection offers much larger space for visualization. The user is not limited to the relatively small vertical field of view as on the Virtual Workbench. When the user zooms in on molecules, he is in fact standing inside or walking around it. This experience cannot be provided at this scale on the Workbench. It seems that the user fascination and presence when steering the proteins in the CAVE is much higher than on the Workbench. The CAVE can also offer a larger immersive environment where a demonstration of interaction with proteins can be given to a larger group of people. We have proved this way that the CAVE is in fact supplementary to the Workbench concept.

However, we have also found significant differences and drawbacks of CAVE applications. One of the problems is much less accurate tracking. On the Workbench the users can accurately perform direct selection and direct manipulation tasks. But in the CAVE the users should rather use remote interaction (ray-casting). Another problem is that the images in the CAVE appear much less focused than on the Workbench.

We have brought into the CAVE a virtual life-size model of the Workbench and we have supposed that MolDRIVE could be used in the same fashion as on the Workbench. User interface and workspace layout remained intact, and in the

same size as on the Workbench. The visual experience in the CAVE that objects appeared less focused but the molecular system could be scaled up so much that on the Workbench it would seriously distort the immersion, as it would simply not fit in. Very confusing in the CAVE was the lack of physical haptic feedback of the Workbench and the Plexipad.

We can conclude from this experiment that the CAVE is better in giving more impressive demonstrations and displaying larger objects. When detail and task accuracy are important, then the Virtual Workbench provides a better environment. It also seems that for everyday laboratory experiments the Virtual Workbench is simply a more realistic solution.

6.2.8 MolDRIVE Performance

As explained earlier in this chapter, the update rate of the simulation and visualization processes are partly independent. The frame rate of the visualization is not directly related to the simulation update rate. However, the simulation is limited to the visualization frame rate. The simulation will not continue until one of the two latest time steps (in the double buffers) has been visualized. To indicate where the performance limitations in the system are, we have conducted some performance measurements. The performance of MolDRIVE has been measured by comparing the most time consuming aspects of our system when performing one time step. These include contributions of the simulation and visualization components and the data transmission through the network. The simulation side of the system is divided in the MD simulation time and the time spent collecting and transforming the requested simulation data (e.g. retrieve particle data and calculate derived grid data). In the visualization, a standard sphere representation (36 triangles) was used for the display of the particles.

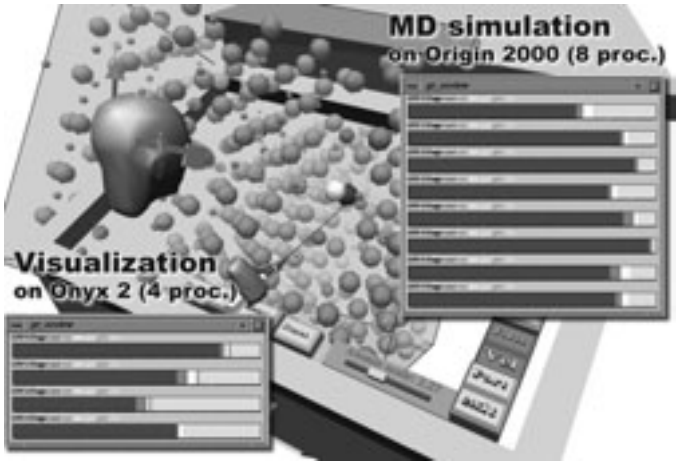


Figure 6.31: Processor load in MolDRIVE: simulation on 8 computing nodes, visualization on 4 processors

In the first measurements of the MolDRIVE system, a DEMMPSI MD simulation of liquid argon of different system sizes was run on 1,2,4 and 8 nodes on the Origin 2000, the best performing supercomputer available to us. The results of these measurements are shown in Figure 6.32.

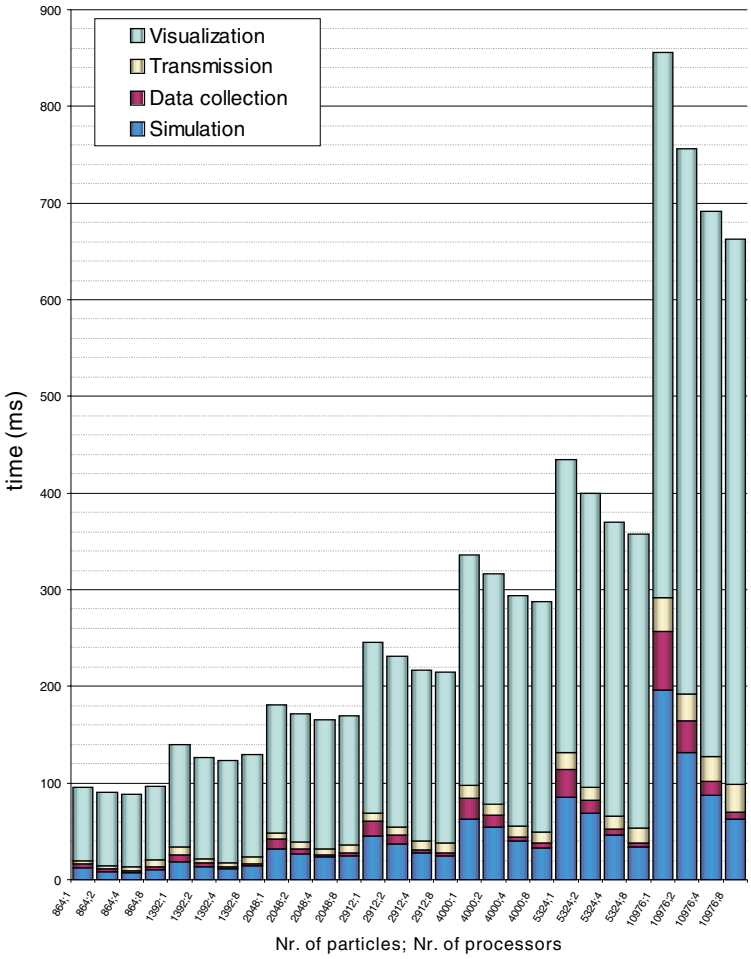


Figure 6.32: MolDRIVE performance: time measurements of a real-time MolDRIVE session on an MD simulation of liquid Argon on 1,2,4 and 8 processors on SGI Origin 2000

It became clear that both the MD simulation and the data collection scaled well. They could produce and transmit data from over 10.000 particles in under 100 ms (more than 10 updates per second) when using all 8 processors. The visualization however did not perform so well. Contrary to our expectations,

it became clear that the overall performance of the simulation, including data collection and transmission, was much better than the visualization. At 2048 particles, the frame rate already dropped below 10 Hz. As expected, the visualization frame rate of the visualization server is independent of the number of processors used for the MD simulation.

However promising the results of the simulation seemed, the good performance was mainly due to the absence of long range particle interaction in the simulation of the liquid argon. These results were compared with measurements on the MD simulation of β-alumina (2088 particles), a more complex simulation which does use long range particle interaction (PPPM). This type of simulation was more representative of the general performance of MD simulations. From the measurements it became clear that the MD simulation can also be the most time consuming component in the system (Figure 6.33).

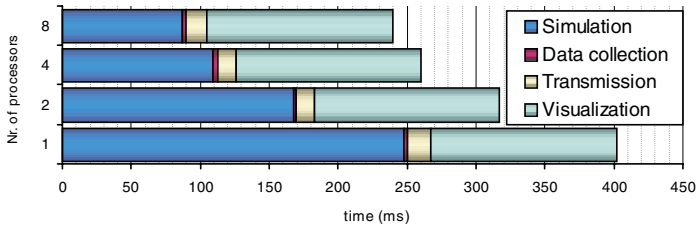


Figure 6.33: MolDRIVE performance: time measurements of a real-time MolDRIVE session on an MD simulation of β-alumina with 2088 particles on 1,2,4,8 processors on SGI Origin 2000

Further it seems that in this configuration, the maximum number of particles in real-time MD simulations is around 2500 particles. Above this number the update rate of the simulation will drop bellow 5 per second (time above 200ms) and the interaction latency becomes too high. The playing back of the recorded simulation timesteps has higher update rates but particle steering interaction is not possible. In playback, the update rate of simulation data is only limited by the hard disk performance (e.g. well over 10.000 particles at 24 updates per second).

Also, the Visualization Client is limited to the display of around 2500 spheres for particles (36 triangles per sphere). However, the performance can be considerably increased by using other particle representations and other visualization techniques. The use of derived data instead of particle data will increase the performance to the limit of 24Hz. Visualization using the data slicer tools does not have a great impact on the performance of the Visualization Client. By the use of the ROI or showing/hiding of all particles of given type, the number of visible particles can be decreased. In this way it is possible to work with a larger simulation system with many more particles, while only few are displayed.

In these measurements, the data collection time and transmission time in real-time simulations appeared not to be of much influence on the total performance of the system. However, we have only used fast grid extraction for derived data and not used any complex feature extraction algorithms. We expect that the data collection can also become a performance limitation when more complex algorithms are used. Moreover, the network transmission was only measured on the high speed Local Area Network. We have done a preliminary test where the simulation runs on a computer at the SARA high performance computing center in Amsterdam while the RWB in Delft was used for visualization. Although we have not performed time measurements, there was no noticeable increase of network delay in the interaction.

In the combination with Gromacs, a small benchmark was performed to indicate possible bottlenecks. Two simulations were compared, one running independently, and one communicating with MolDRIVE. For a small protein system (around 500 particles), the data transmission time turned out to be very long compared to the time of a single integration step. In this case, the communication overhead resulted in an increase of the simulation update time of over 200 percent. We expect that this overhead will diminish when using larger and more complex simulations. A solution to the influence of communication overhead could be to dynamically adjust the number of integration time steps before transferring the data. Instead of sending the simulation data of every time step, more time steps can be calculated before one is sent to the visualization server. Although we have not conducted more time measurements on the Gromacs performance, it became clear that it outperforms DEMMPSI when only considering the number of particles. We expect that Gromacs will be able to simulate almost 10.000 particles in under 100ms per time step (above 10 updates per second).

As expected, running the simulation remotely on a supercomputer turns out to be a big improvement of simulation performance compared to the locally running MD simulation on the visualization server as in Vrsim. The network transmission from the Simulation Nodes to the local machine takes little time compared to computing a simulation time step for larger simulations, so the improvement remains significant. Moreover, the Onyx2 is relieved from the burden of a computationally intensive MD simulation, leaving more resources for the Visualization Client. Although the capacity of the visualization server has improved, it can still become a bottleneck of the entire system when displaying too many individual particles. The use of derived data and other visualization tools such as ROI and particle type selection reduce the number of particles to be drawn, making it possible to deal with larger systems.

We have seen that the bottleneck of the system shifts between the simulation, the visualization, and the network transfer. The limitation of the simulation (amount of particles, simulation program, complexity) cannot simply be overcome by techniques other than simulation recording and playback. The performance limitations of the Visualization Client can be stretched to the hardware optimum of 24 Hz, by selecting a fast visual representation of particles.

6.2.9 MolDRIVE: Summary and Discussion

After designing of the MolDRIVE architecture we started to implement most of the basic techniques such as network communication, shared memory, and basic visualization and simulation interface. In addition we learned from the previous project (Vrsim) and early tests. Surprisingly, we completed our first working prototype earlier than expected. Because of the flexible combination of the RWB Library and Performer also the development of visualization techniques and interaction methods was more effective. Although the Onyx2 is a high performance graphics workstation, the real graphics performance was below our expectations (It will be always possible to generate a very complex geometry, which cannot be rendered at interactive frame rates). Nevertheless, by the use of derived data and a new data representation we were able to develop an interactive virtual environment, which is well suited for the visualization and computational steering of many MD applications. MolDRIVE works on the Responsive Workbench as well as in the CAVE.

The overall performance of MolDRIVE shows that is it now possible to perform real-time MD simulations of up to 2500 particles and visualize it interactively. The versatility of the system is illustrated by several case studies that have been performed with the system. The 3D visualization techniques revealed spatial configurations and important structures in the materials studied. In combination with particle steering this allows researchers to conduct experiments and analysis on a nano-scale. Especially the integration with the Gromacs MD software, which enables the steering of proteins, has been successful. The promising results in combining Gromacs and MolDRIVE have led to the continuation of using this system for molecular modelling. Furthermore, MolDRIVE will be integrated with VRX, and a more effective visualization environment may be expected.

Our system may be of certain importance in research, for example on material properties and molecular modeling. It must be said that MolDRIVE sessions with real-time simulations are not suitable for many other MD applications. During our project, it became clear that most MD users use very large and very slow simulations. Only after weeks or months they start with the actual analysis of their simulation results. As our system is only able to process small scale simulations in real time, the large simulations are not suitable for real-time use with MolDRIVE. However, these MD simulations can be simulated off-line using our system and intermediate results can be visualized on demand.

6.3 Visualization of Cumulus Clouds

6.3.1 Introduction to Atmospheric Simulations

Transport processes in the lowest kilometers of the atmosphere (boundary layer) are very complicated due to the turbulent nature of the air motion [Stull, 1988]. The length- and time-scale of the turbulent motions is ranging from millimeters to kilometers and from fractions of seconds to hours. These turbulent motions (eddies) are very important since they are responsible for the vertical transport of heat, moisture (latent heat), and pollutants.

The presence of clouds in the boundary layer makes the dynamics even richer. It forms also an additional complication due to the phase changes (condensation/evaporation) and the impact of clouds on short- and long-wave radiation. The understanding of the turbulent dynamics in the lowest parts of the atmosphere can be increased by observational or computational methods. The observational methods include e.g. aircraft measurements, balloon measurements, lidar/radar measurements, and satellite imaging. As the observational techniques do not provide a complete 3D state of the boundary layer and its evolution in time, the numerical techniques have become a good alternative for studying atmospheric boundary layers and Large Eddy Simulations (LES) are usually used for these purposes. The idea behind LES is to fully resolve the large scale turbulent motions in space and time, and to parameterize (i.e. model) the smaller eddies. The concept of LES is rooted in the assumption that the largest eddies are the most energetic and therefore responsible for most of the turbulent transport. Many comparisons of LES results with observational data have revealed the capability of LES to mimic atmospheric boundary layers, not only for dry situations, but also for the even more complicated cases where clouds (cumulus, stratocumulus) are present.

The advantage of numerical data is clear: we obtain the full 3D state of the key quantities (momentum, pressure, temperature, moisture) as well as their evolution in time. The problem with this may be equally clear: the amount of data is huge, and has to be filtered and post-processed for further study. An often used method is to calculate spatial/temporal averages of the data and study the average quantities instead. Such data reduction may be suitable for many purposes, but it must be realized that important information with fine details of the data is lost.

For example, in a recent study [Siebesma & Jonker, 2000] a cumulus topped boundary layer was investigated by means of LES with particular focus on cloud geometry. The boundaries of the numerical clouds revealed an anomalous scaling behaviour with a fractal dimension equal to the value known from observations [Lovejoy *et al.*, 2000]. It is this kind of information that is lost when only statistical averages are considered. This study shows that geometrical features of cumulus clouds are nicely represented in the numerical model, and that the LES is a good numerical method for simulation of cumulus clouds.

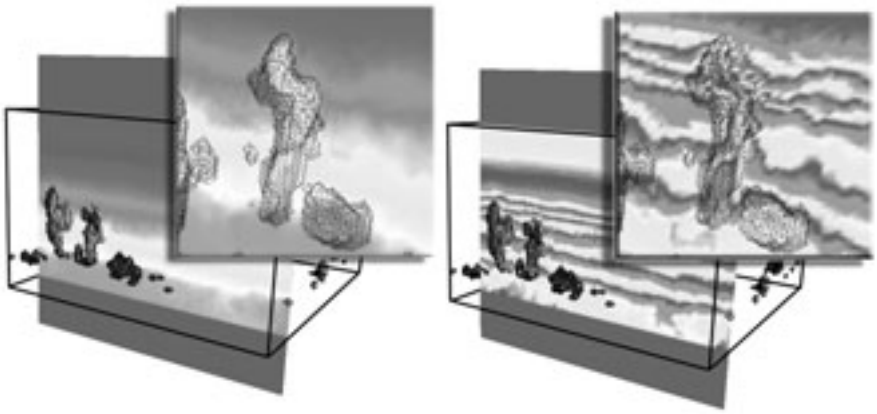


Figure 6.34: Informal visualization of the cloud field data using AVS; wireframe iso-surfaces combined with a vertical color slicer; different color mappings compared: non-periodic color spectrum (left), periodic color spectrum (right);

However, we have to find ways how to efficiently explore the large datasets obtained from LES. Virtual environments offer the users the opportunity to investigate the time evolving datasets in a computer-generated 3D environment with natural forms of interaction. Our experiments with VRX on the Virtual Workbench will demonstrate how evolving 3D cumulus cloud fields can be visualized, enabling an observer to identify clouds that satisfy a number of basic requirements (large enough size, undergoing the full life cycle from cloud birth to decay, etc). By interactively defining a region of interest (in space and time), the user will explore the evolution of all relevant variables, such as moisture, temperature, pressure, and the 3D velocity vectors. We will adapt visualization techniques for VR: i.e. streamlines, particle tracing, releasing of virtual dust, and simultaneous visualization of scalar and vector fields.

The goal is to get a better understanding of the spatial/temporal structures in atmospheric boundary layers with cumulus clouds. By enabling the user to freely maneuver (navigate) in the dataset, selecting interesting regions and events, and visualizing the relevant quantities on demand, better insight will be gained into the complex flow around and below clouds. Two specific questions that will be addressed:

- Observational atmospheric studies have revealed a thin shell of descending air around clouds. The origin of the descending motion however is still in discussion. The opposing hypotheses are: a) mechanical forcing and b) evaporative cooling due to mixing of environmental air with cloudy air. By employing the Virtual Workbench we aim to extract from the data whether or not the LES clouds indeed possess this shell of descending air and, if so, to identify the origin of the phenomenon.

- There is a large debate in the atmospheric community on the dominant mixing process in clouds. The mixing process deals with a transport of moisture into and outside the cumulus cloud [Stull, 1988]. Two hypotheses exist: a *dominant lateral mixing* (moisture is mainly lost at the sides of the cloud) versus a *dominant vertical mixing* (moisture is mainly transported from the bottom and is lost on the top of the cloud). An interactive exploration of the cloud field dataset on the Virtual Workbench should gain more insight of the mixing processes.

6.3.2 Cloud Visualization in VR

Atmospheric simulations are usually complex and computationally intensive, and they deliver large time-dependent data sets. The LES used in this case study provides a large dataset (20 GB uncompressed, 600 time-steps). The main physical quantities are momentum, pressure, temperature and moisture. For Virtual Reality it is a real challenge to be able to interactively visualize and browse through such a large time-dependent dataset. In the worst case, when the user wants to visualize all available data components, the system would have to read from hard-disk storage about 30MB of data per time step. Without a suitable compression scheme and hierarchical level of detail representation of data it will not be possible to interactively access and explore such a large dataset. The user is searching for an interesting cloud with a complete life cycle inside the simulated time interval. The spatial relations and the simulated physical properties around the selected cloud have to be explored in detail.

Initially, we have visualized the cloud data in AVS, see Figure 6.34. The images show geometries of the clouds (iso-surfaces of moisture) together with temperature. Two different color mappings are compared. From our experience we have concluded that a desktop visualization cannot offer a really interactive 3D exploration of the full time-dependent data. Usually, the data must be processed off-line with a suitable visualization technique to produce an animation, and then the data flow in time can be analyzed. The existing desktop visualization systems (like AVS, Open DX, or VTK) are very good in static and animated visualizations. Unfortunately, these systems do not have a suitable VR interface. We have learned from the desktop visualization and adapted some techniques for VR. We have designed and implemented an interactive and responsive visualization system for large time-dependent atmospheric data containing cumulus clouds. The main requirements of this case study were:

- Maintaining random access to any time step of data
- Pre-processing schemes of the data to speed up the visualization.
- Automatic selection, clustering and attribute calculation of clouds.
- Automatic tracking of clouds through the whole simulation.
- Spatial/temporal and content-based selections (like a region of interest in space and time; respectively a cloud of interest and its path in time).

- Interactive 3D playback of cloud simulation with the ability to quickly (interactively) explore the simulated data.
- Application of exploration and visualization tools, described in Section 5.2.

Related Work on Atmospheric Visualization in VR

Data from numerical simulations and measurements of atmospheric processes have been traditionally visualized in existing universal visualizations systems such as: AVS, Iris Explorer, OpenDX, and VTK. Also specialized programs for visualization of the atmospheric data were developed, i.e. Vis5D [Hibbard *et al.*, 1996; Hibbard & Santek, 1990].

Due to the 3D nature and complexity of this data, the interactive exploration in virtual environments should help enormously in studying of the processes in the atmosphere. Some visualization systems that were originally designed for desktop systems have been adapted for VR usage in CAVEs, workbenches, panoramic walls, or HMDs.

One good example can be adaptation of Vis5D for the CAVE in the Cave5D system, see Figure 2.17. The Cave5D framework [Wheless *et al.*, 1998] integrates the CAVE Library with the Vis5D library in order to interactively visualize atmospheric data in the Vis5D file format on the ImmersaDesk or in the CAVE.

However, Vis5D and Cave5D deal with another type of spatial domain of the data and therefore another level of detail of the visualization techniques is required. Vis5D was designed for visualization of global weather and atmospheric conditions in areas of several thousands of square kilometers (simulation timesteps in the order of minutes or hours), where the scientists are studying only global effects in the atmosphere. The scale of related simulations is very different to the cumulus clouds case study (area of 6x6 km, timesteps in the order of seconds).

In this case study we demonstrate a detailed visualization and exploration of pre-simulated time evolving cloud field on the Virtual Workbench.

6.3.3 Data Management and Analysis

For use of the simulation data in our visualization application the data had to be pre-processed. The Large Eddy Simulation (LES) produced a large time-dependent dataset, with grid dimensions of 128x128x80 and 600 time-steps, which represents one hour of simulated clouds in an area of 6x6x3 km. Per grid cell the following quantities have been computed: the air velocity vector (u, v, w) and several scalars: meteorological temperature θ_l, total water q_t and liquid water q_l. The data components were stored in 6 separate files. Originally they were double precision floats, which meant 64 bits per scalar. One data component file was thus 128x128x80x8 bytes (10 MB). To manage the enormous data volume we had to compress the data into 16-bit integers. Not to lose the 64-bit

float precision on a given range we have used a *float-to-integer* ratio. This way one data component file needed only 2.5 MB of harddisk space. After reading the file into memory, the integers (16 bits) must be converted back to floats using an *integer-to-float* ratio ($q_t, q_l \approx 10^{-5}$, $\theta_l, uu, vv, ww \approx 10^{-3}$).

Optionally, we can use Reynolds' decomposition to compensate the influence of the average wind speed in the data.

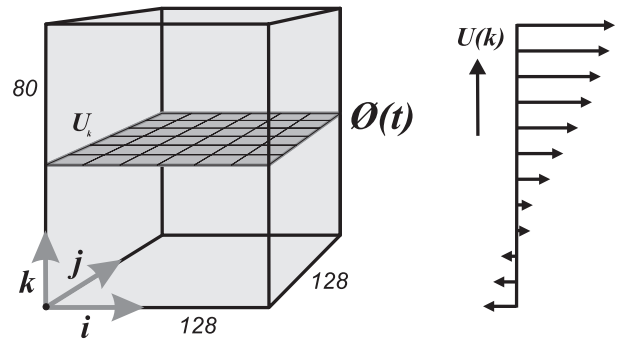


Figure 6.35: Reynolds' decomposition of a dataset; determining the average $\mathbf{U_k}$ in time of all scalars in a plane of given height

$$\mathbf{U_k} = \tfrac{1}{N^2T}\textstyle\sum_{i,j,t} u(i,j,k,t) \qquad i,j = 1..128, k = 1..80, t = 1..600$$

$$\mathbf{V_k} = \tfrac{1}{N^2T}\textstyle\sum_{i,j,t} v(i,j,k,t) \qquad \mathbf{W_k} = \dfrac{1}{N^2T}\displaystyle\sum_{i,j,t} w(i,j,k,t)$$

$$\boldsymbol{\theta_{lk}} = \tfrac{1}{N^2T}\textstyle\sum_{i,j,t} \theta_l(i,j,k,t) \qquad \mathbf{Q_{tk}} = \dfrac{1}{N^2T}\displaystyle\sum_{i,j,t} q_t(i,j,k,t)$$

From scalar components (u, v, w, θ_l, q_t) we subtract the plane-average in time according to the height k ($\mathbf{U_{[k=1..80]}}$,$\mathbf{V_k}$, $\mathbf{W_k}$, $\boldsymbol{\theta_{lk}}$, $\mathbf{Q_{tk}}$); $N = 128, T = 600$.

$$U\prime(i,j,k,t) = u(i,j,k,t) - \mathbf{U_k}$$
$$V\prime(i,j,k,t) = v(i,j,k,t) - \mathbf{V_k} \qquad W\prime(i,j,k,t) = w(i,j,k,t) - \mathbf{W_k}$$
$$\theta_l\prime(i,j,k,t) = \theta_l(i,j,k,t) - \boldsymbol{\theta_{lk}} \qquad Q_t\prime(i,j,k,t) = q_t(i,j,k,t) - \mathbf{Q_{tk}}$$

This way only data fluctuations can be made visible, which has been found very helpful for studying time-dependent data in detail; the average wind speed has been filtered out.

6.3.4 Visualization of Cloud Geometry

Geometric objects were extracted for each time step to create a natural visual representation of the clouds. For this, iso-surfaces were generated from the q_t data files (liquid water). The geometries were smoothed and the quality of triangulation was optimized. The resulting cloud geometries were stored in Performer binary files (PFB), see Figure 6.36. These binary geometrical files (about 1,5MB each) could be read very rapidly and inserted directly into the Performer scene graph. In the visualization application these geometric objects could be placed in the data space, where the other cloud data are visualized.

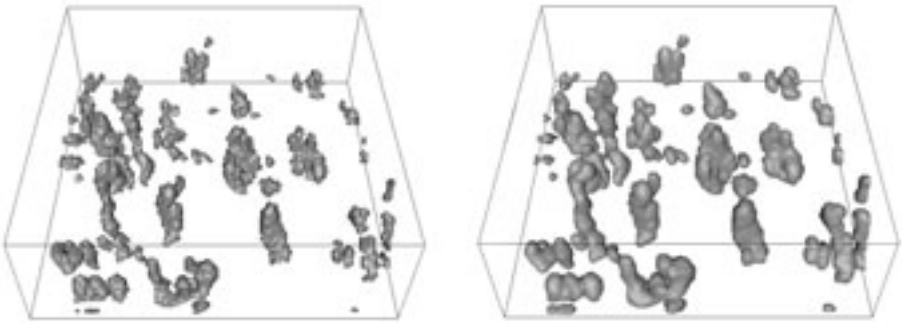


Figure 6.36: Cloud geometry is based on the liquid water data: raw iso-surfaces (left) and smoothed and optimized surfaces (right)

It is obvious that the iso-surface representation produced by *VTK* is far from a natural-looking smooth cloud representation. However, our motivation was not to implement realistic cloud visualization. As we are dealing with continuous LES data we also have to visualize continuous and smooth iso-surfaces. Due to the rather small grid dimensions (128x128x80) of the simulation the effects of point sampling and aliasing are reflected in the quality of the iso-surfaces. To cope with this, and generate "nicer" cloud geometry we have used a Gaussian smoothing filter before using the Marching Cubes algorithm for iso-surfaces. Result is shown in Figure 6.36 (right).

The Marching Cubes algorithm as such does not produce high quality triangular meshes [de Bruin *et al.*, 2000]. Although we have smoothed the data with a Gaussian filter, the simulation grid is still clearly visible in the mesh (Figures 6.37a,d). Also the quality of the triangulation is quite low. For an optimal rendering of the triangular mesh the triangles should be as close as possible equilateral. We have used the *vtk triangular mesh smoothing filter* for improvement (Figures 6.37b,e). This final cloud representation is stored in a PFB (Performer Binary) geometry file. Data consisting of 10-50 clouds with about 40-60 thousand triangles is stored in a binary geometry file of 1-1.5 MB. The amount of

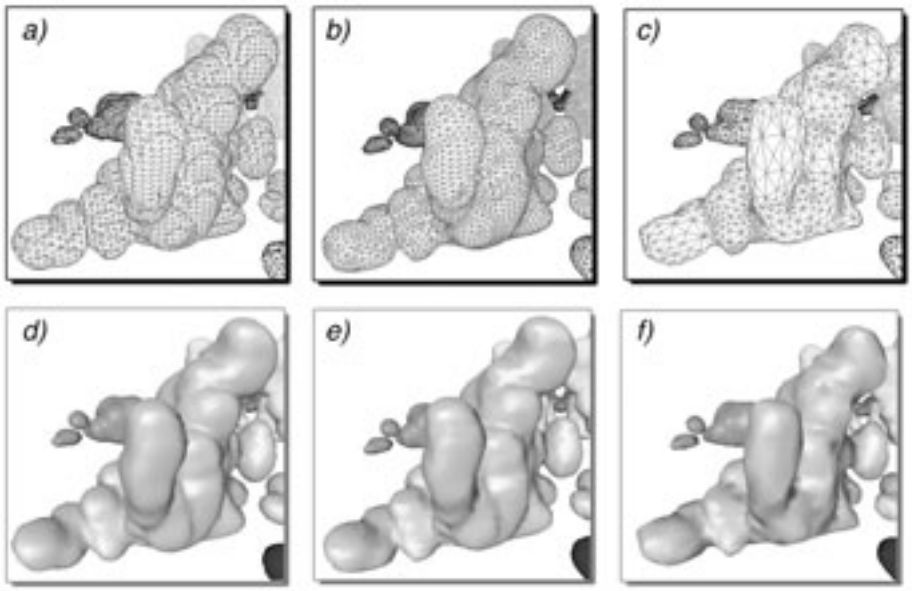


Figure 6.37: Cloud geometry refinements: a,b,c) in wire-frame, d,e,f) shaded; a,d) - original iso-surface mesh (12 clouds) produced by Marching Cubes (49.000 tris, 1.6MB); b,e) - optimization and mesh smoothing (46.000 tris, 1.5MB); c,f) - mesh decimation/simplification (18.000 tris, 600kB)

triangles and the file size can have significant impact on the performance of our visualization system especially, when the user is playing back the simulation on the Virtual Workbench, see Figure 6.44. One of the bottlenecks is the rendering pipeline. Our OpenGL graphics hardware can process and render theoretically up to about 10 millions of triangles per second (but practically 5-6 mil.). For the frame rate of 2*25 Hz (in stereo) our scene may contain about 100-200 thousand of triangles. As our visualization system also has to display other things than the cloud geometry, it is reasonable to use a suitable mesh simplification. We have applied *vtk-decimation*, see Figures 6.37c,f. The cloud geometry can be decimated to 18 thousand triangles, which needs only 600 KB. The whole dataset (600 time-steps) was pre-processed and the cloud geometries were generated in both versions: not-simplified and simplified. This data pre-processing took 10 hours. Each cloud has been processed separately in the given time step.

The geometries of all clouds in one time-step were stored into a single binary file (PFB). Each cloud can be identified by a cluster number and has its own pfGeode in the scene graph sub-tree. After clustering (Section 6.3.5 of the cloud dataset, at each time-step a different cluster number is assigned to each cloud. The correspondence between the clouds in consecutive time-steps has to be determined by a tracking routine. Clouds of one time-step are read at once and any of the clouds can be selected or colored, see Figure 6.38.

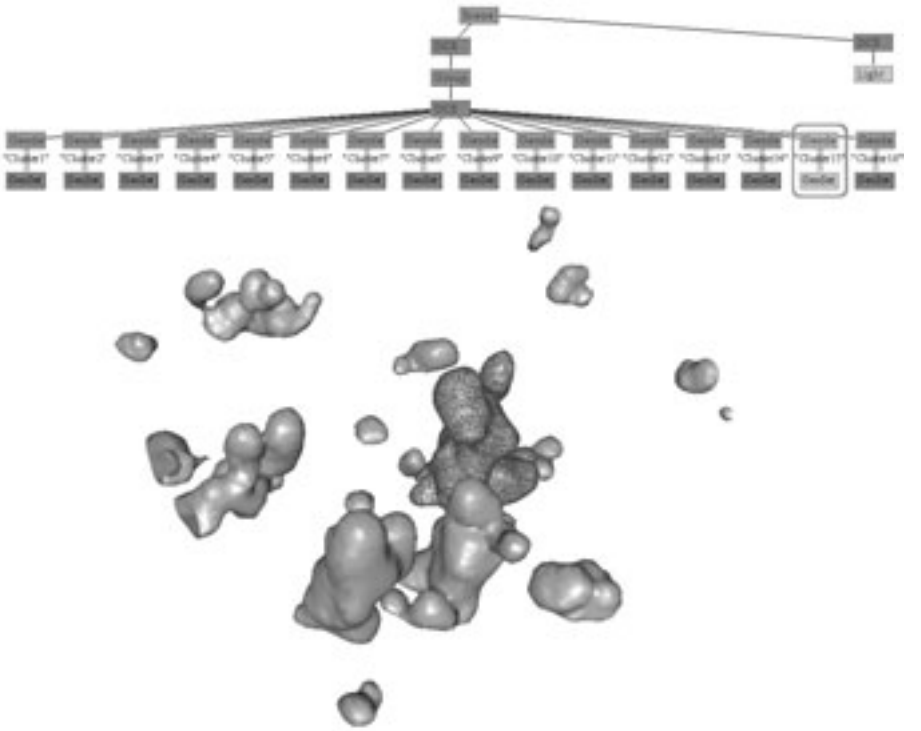


Figure 6.38: Performer scene graph of the cloud geometry; pfDCS (Performer Dynamic coordinate system) contains separate cloud clusters in each pfGeode node, which can be identified by a name and number for each cluster. Each pfGeode contains pfGeoSet with the geometry.

When our system has to read only the cloud geometries (about 1.5MB per file and time-step) from the harddisk and display about 40 thousand triangles, we can still achieve a high rendering frame rate of more than 24 Hz and data update rate of 10 Hz. But as soon as additional data components have to be read from the harddisk and visualized simultaneously, the update rate drops drastically (< 1 Hz). From our measurements it is clear that we cannot expect much more than 10-15 MB per second (continuous) from our data storage system. In case of reading all five data components (the liquid water is not needed, we use the cloud geometry instead) the data amount would be: 5x2,5MB + 1,5MB = 14MB; the data update rate about 1 Hz. Conversion of integers to floats costs only about a milisecond. For performance reasons it is better not to overload the data input when the user is performing interactive exploration of the data in time. It makes sense to read the cloud geometry (1-1.5MB) together with one data component (2.5MB). The data update rate would be then about 3 Hz. The size of the data is not a problem when exploring a single time-step. It becomes a bottleneck only when the user wants to watch the simulation data in playback mode.

Practically, there are two solutions. The first solution is to sub-sample the dataset into a smaller grid, for example 32x32x20 x2 bytes, which needs only 40kB. The whole dataset could be pre-processed and a sub-sampled version of the data could be used for interactive previews of the simulation. A severe disadvantage of this method is that it removes many interesting details of the data, but an advantage is that it preserves a global overview.

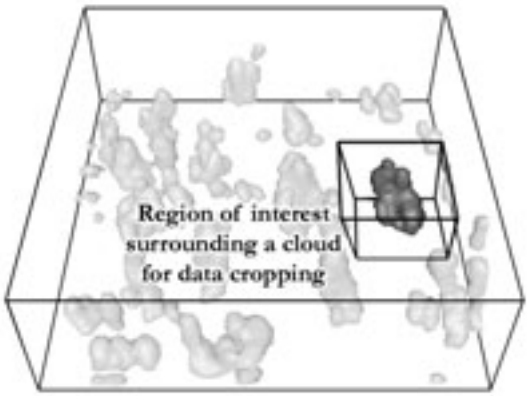


Figure 6.39: Feature-based cropping of the cloud data

Another, and much better solution, is a feature-based cropping of the data. We know that the user is interested in a small region of the data that surrounds a chosen cloud formation, see Figure 6.39. When our system could track the cloud in time, we could read only the data around this cloud, which would be just a fraction of the whole data. The diameter of the region of interest can be determined as maximal cloud size over the time series + some margin. After a selection of the cloud, the whole dataset and corresponding clouds would be processed and cropped. An advantage of this method is that no details would be missed, but context will be lost. To cope with losing the context we can combine both methods: global view on the cloud dataset in low resolution, and local high resolution around a chosen cloud. Technical problem is only finding a good data structure for browsing in space/time and resolution.

6.3.5 Clustering and Tracking of Clouds

The q_l data (liquid water) contain non-zero values where the visible clouds are, and zero values everywhere else. It is thus quite simple to make data selections. The next step is to find the clusters of neighboring cells that satisfy the selection criterion. The process of finding these data clusters and assigning identification numbers is called clustering. For data clustering we have used a vertex correspondence, see Figure 6.40. As we are interested only in sufficiently large clouds, we ignore smaller clusters with less than 100 data cells. To each clus-

ter of such data a cluster number is assigned. A single cluster corresponds to a single cloud in the cloud field.

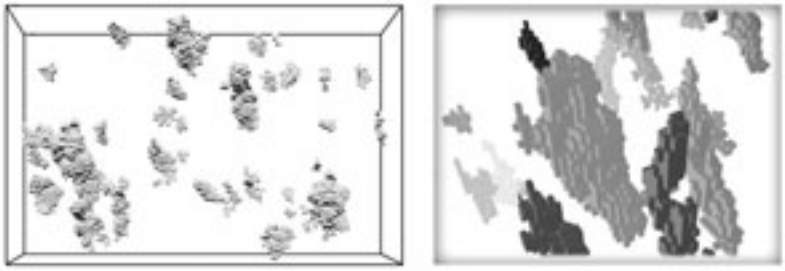


Figure 6.40: Selection of cloud data (left); clustering of clouds (right)

An optimized iso-surface is generated for each cloud separately. All cloud geometries of one data-frame are stored into one binary geometrical file (PFB). The whole dataset (600 time-steps) is processed frame by frame. The pre-processing stage generates a series of files containing cluster attributes (center position, dimensions, volume, bounding box), and cloud geometries.

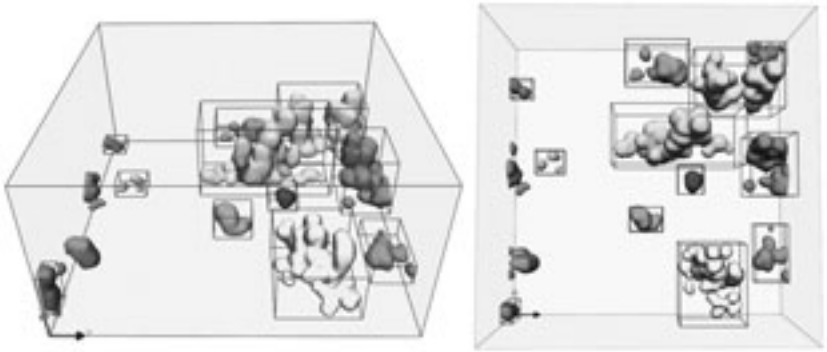


Figure 6.41: Coloring of cloud clusters: the clustering method also supports periodic boundaries (see also the Color Section).

After clustering and generating cloud geometries we perform time-tracking of the clouds. We have applied a user assisted tracking scheme [Brinckman, 2002], where an algorithm tries to find the cloud cluster correspondences between consecutive time-steps. After the automatic tracking the user can edit the correspondences manually and restart the automatic tracking from a given time-step, see Figure 6.42. Some basic events can eventually be detected, like cluster birth, death, join and split [Reinders, 2001]. The main goal of our simple cloud tracking scheme is to provide the user with a cloud selection through the whole life-cycle of a chosen cloud, enabling the user to automatically highlight one selected cloud or hide the other clouds.

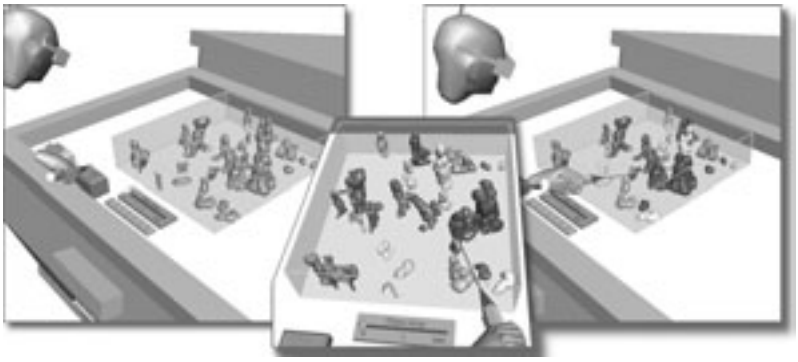


Figure 6.42: User-assisted tracking on the Virtual Workbench (see also the Color Section)

The cluster correspondence is made visible by assigning the same color to corresponding clouds in different time-steps. We have tested several schemes for assigning colors to the cloud clusters, see Figure 6.43. The simple scheme (Figure 6.43 - top) does not provide enough difference between the colors. However, in the optimized scheme (Figure 6.43 - bottom) the colors are sufficiently distinct in the HSI color space (hue, saturation, intensity), so that the user can always distinguish between two different clouds (clusters). Due limitations of the VR displays and the human eyes, we can use 100% on the hue scale, 75% on the saturation scale, and only 60% on the intensity scale.

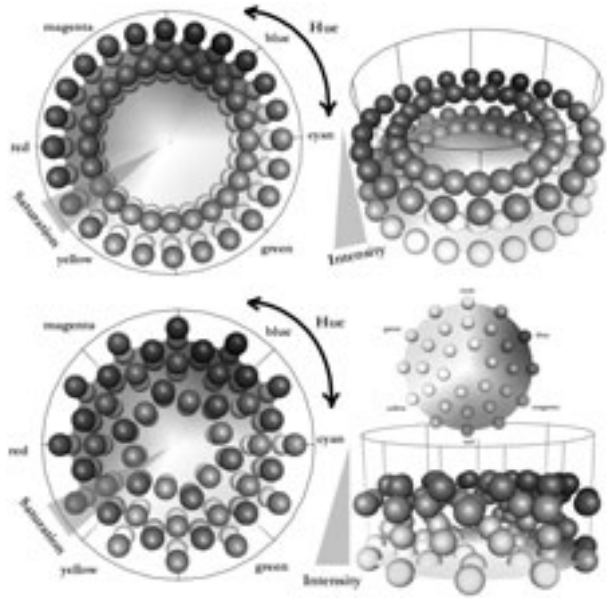


Figure 6.43: Simple color scheme (top) and optimized color scheme (bottom) with 96 different colors in the HSI color space (see also the Color Section)

6.3.6 Interactive Exploration and Visualization of Cloud Data

The exploration process begins with a quick search through the dataset, see Figure 6.44. The Plexipad is holding a time-control widget for navigation in the time of the simulation. The stylus can be used for operating the widgets, as well as for navigation in the visualization of the cloud field. The clouds are tracked in time and corresponding clouds in subsequent frames are assigned the same color. The user can select a cloud with interesting properties and complete lifecycle.

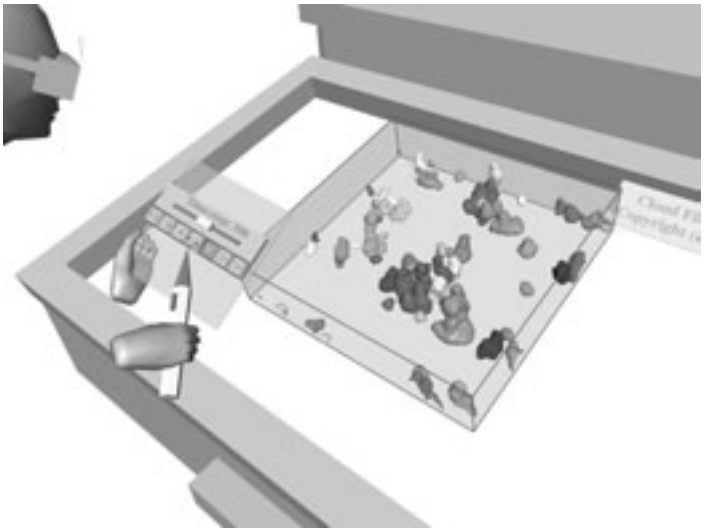


Figure 6.44: Playback of the cloud field simulation. Clouds are tracked in time and assigned with color to show the correspondence in time. The user is searching for a cloud with interesting properties.

The direct data slicer can be attached to the Plexipad, allowing highly interactive exploration of the data. The stylus can be used to probe the data on the surface of the direct data slicer. While using the direct data slicer, the stylus can be used to operate the rest of the virtual environment, using the 3D GUI to change for example the data visualized or adjusting the color mapper, see Figure 6.45. A color mapper widget can also be attached to the data slicer, enabling the color mapping adjustments without changing view context, which has been found very convenient.

One of the visualization techniques that we tried in VR was a direct volume rendering [Meissner *et al.*, 2000]. A ray-cast volume rendering is too slow for VR. Therefore we have tried to use point primitives to emulate volume rendering, see Section 5.2. The preliminary results show a large potential of the point rendering techniques. It is very fast, 50 - 100 thousands points can be drawn without any drops in frame-rate.

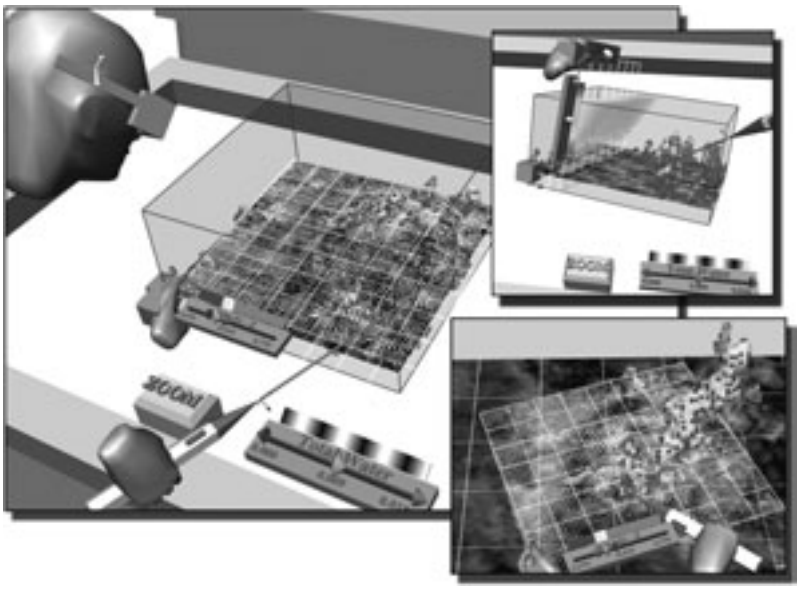


Figure 6.45: The direct vector data slicer is showing air velocity vectors colored with vertical velocity. Point-based volume rendering shows the q_l data (liquid water). The horizontal data slicer (gray-scale color mapper) shows the q_t data (total water). On this slicer constraints are applied so that it can move only in the Z direction, while the vector slicer can freely move with the Plexipad (see also the Color Section).

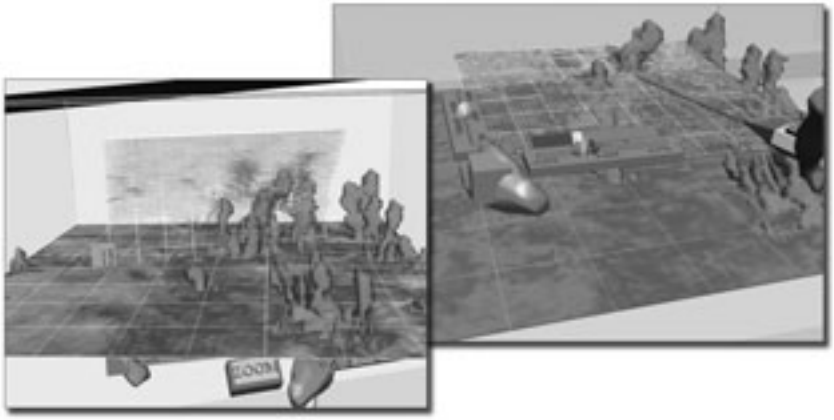


Figure 6.46: Plexipad contains a direct vector data slicer, showing air velocity vectors colored with vertical velocity. The iso-surfaces show the cloud volumes (see also the Color Section).

We decided not to use the *vtk iso-surface* for dynamic visualization of the cloud geometry (as shown in Figure 6.46) in the exploration sessions on the RWB because an iso-surface generation takes even on our high performance visualization server more than a second for a dataset of these dimensions. Moreover, the standard iso-surface routine does not produce sufficiently nice and optimal triangular meshes. Therefore, we use the pre-processed and optimized cloud geometries as described before.

Cloud Exploration

Important effects also happen inside the clouds. Therefore we had to develop an intuitive and functional way of cloud-interior exploration. Transparent surfaces do not work well in stereo. We designed and implemented Plexipad clipping technique. The Plexipad can be used to naturally define one clipping plane, see Figure 6.47. The user can naturally maneuver the clipping plane through the cloud geometry. For awareness of what is being clipped and clarity of the clipping technique we have added a wire-frame of the rest of the clipped cloud. For convenience the wire-frame display can be disabled and hidden.

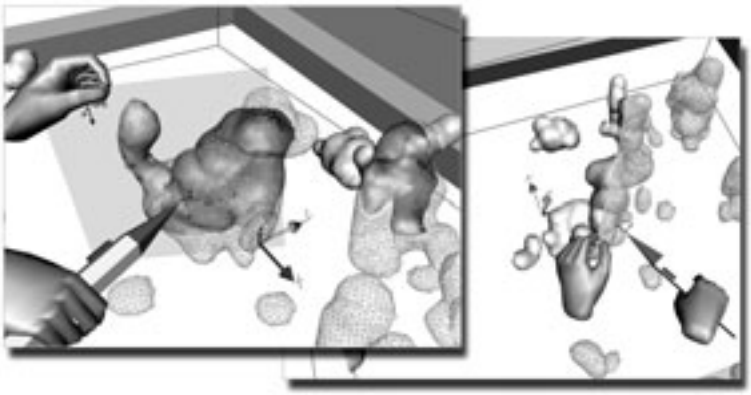


Figure 6.47: Clipping of cloud geometries with Plexipad: one clipping plane

The Plexipad clipping technique is based on OpenGL and the geometric primitives are clipped in the graphics hardware and not in software parts of the graphic pipeline. OpenGL thus has to obtain a complete set of geometric primitives (triangles, lines) with up to 6 clipping planes. What remains after the clipping with the half-spaces is rendered on the screen.

An advantage of the hardware polygonal clipping is that it is very fast. A disadvantage is that it does not allow you to display the removed parts. Therefore we have to send the cloud geometry into OpenGL twice with opposite orientation of the clipping plane to get the wire-frame geometry. It puts an extra load on the graphic performance. When the cloud geometry has too many tri-

angles, a drop in the frame-rate can be seen. In such cases it is better to use the decimated version of the cloud geometry.

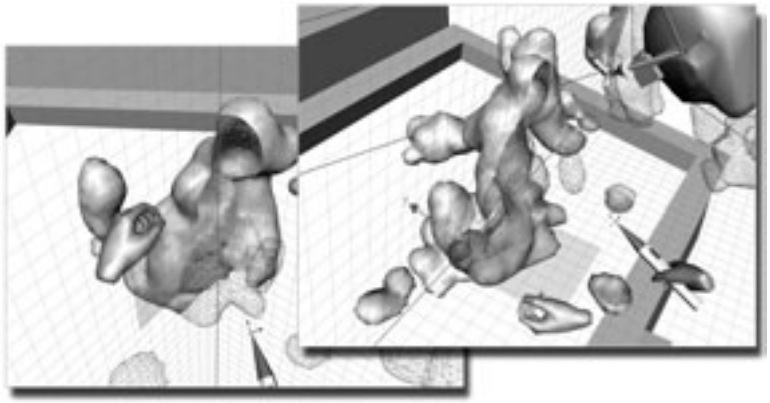


Figure 6.48: Clipping of cloud geometries with Plexipad: two clipping planes

Clipping of cloud geometries with Plexipad opens an inside view into the cloud for detailed visualization. We have found that especially for cloud objects it is better to use two clipping planes (horizontal and vertical), as shown in Figure 6.48. In this way the user has a better view on the cloud in a vertical slice and the vertical relationship is clearly interpreted. The vertical clipping plane is aligned with the edge of the Plexipad. For clarification the reference grid of the clipping planes can be displayed, otherwise the clipping may confuse the user and it is not clear which plane is doing the clipping of the geometry. The clipping technique is fully configurable *(1 or 2 clipping plane(s), wire-frame ON/OFF, clipping-planes reference grid ON/OFF)*.

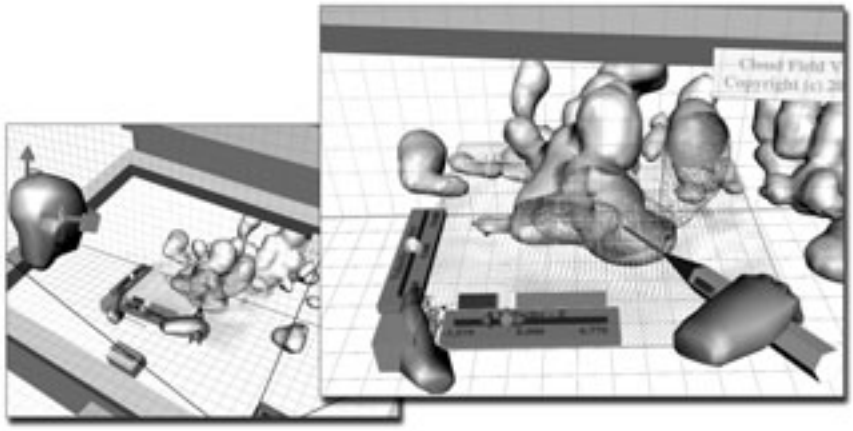


Figure 6.49: Plexipad has attached two clipping planes and controls the vector data slicer. Velocity vectors (u, v, w) are visualized in a given slice. Vertical velocity w is mapped onto the vector's color (see also the Color Section).

Moreover, the Plexipad can be used at the same time for visualization of the data in a given slice of the data space. Figure 6.49 shows an interactive exploration of the air velocity (u, v, w) inside and outside of the cloud. The color mapper and the length of the vectors can be adjusted by the control widgets on the Plexipad.

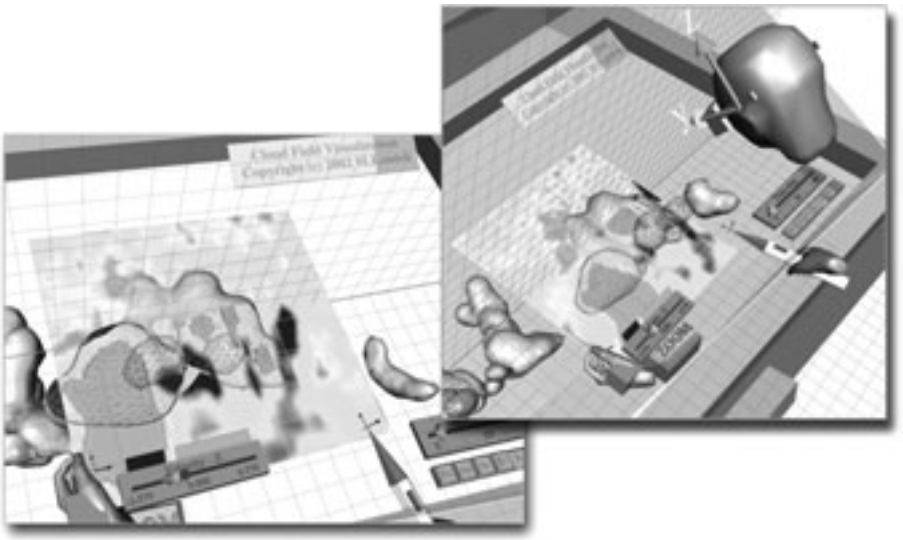


Figure 6.50: Plexipad with 2 clipping planes and direct data slicer that is showing the vertical velocity w (see also the Color Section).

Figures 6.50 and 6.54 show a user exploring the vertical velocity in the cloud with the direct data slicer. This way the user can study in detail the mixing process inside the cloud. In the figures it is clearly visible that the air comes into the cloud from the bottom and flows upwards inside. On the boundaries of the cloud the air flows downward. In other visualization with the temperature data θ_l we have seen more complicated and turbulent behaviour. But a general trend seems to be that the warm moist air enters the cloud from beneath, while it cools down on the sides and on the top of the cloud and moves down.

Color Mapping

A well-fitting color mapper is of a great importance for the data slicers. We have tested several color mapping schemes on the cloud data, see Figure 6.51. Each of the color mappers offers a different look on the data.

The color spectrum and the gray-scale work well in horizontal slices of the data, when they are calibrated with the mapping slider widget to get enough contrast in the textured slices, see Figure 6.51 a,c).

An advantage of these direct color mappings is the visual uniqueness when one data value projects onto one color. Disadvantage is that small gradients in data are less visible. To facilitate this we have designed periodic color mapping schemes (see also Section 5.2.3) with user-specified number of color cycles, see Figure 6.51 b,d). In the periodic color mapping the gradients are visible, but one color corresponds with several data values.

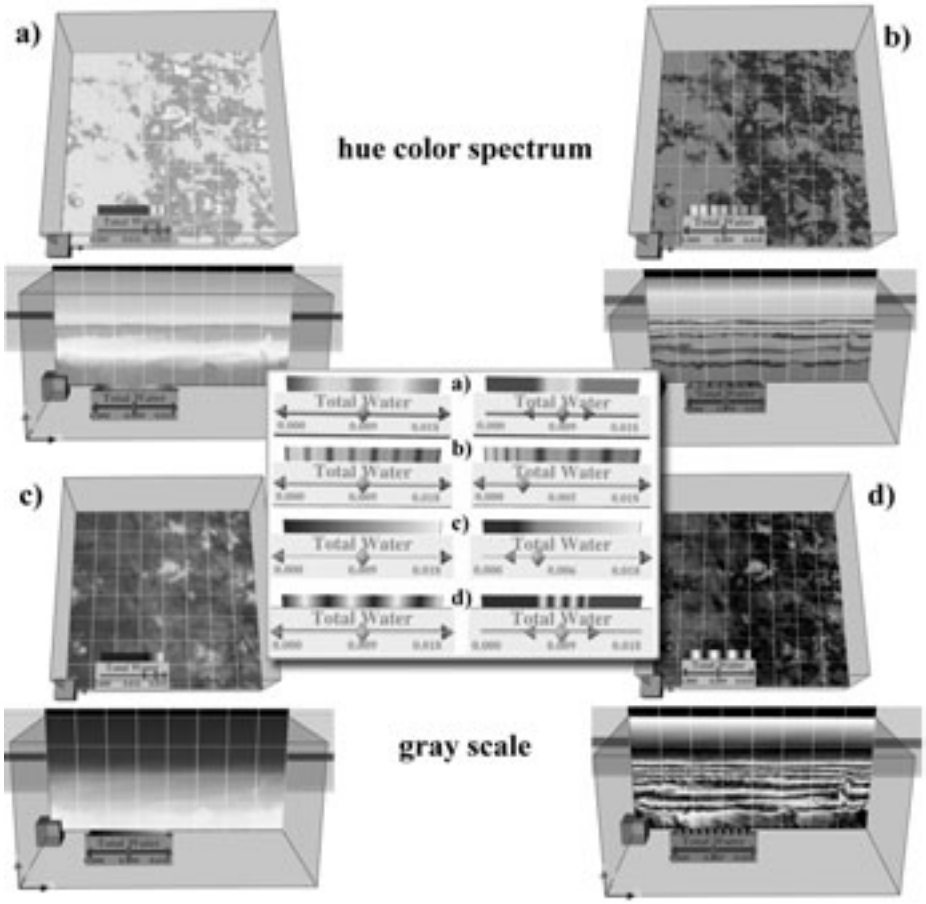


Figure 6.51: Color mapping scenarios: rainbow spectrum (a), periodic rainbow spectrum (b), gray scale spectrum (c), periodic gray scale spectrum (d); (see also the Color Section)

Interactive Streamlines

For visualization of the air flow we have implemented a virtual exploration tool with interactively generated streamlines. It uses the *vtk-streamline* generator.

Using multi-threading, it utilizes the processing power of the SGI Onyx 2 for integration of the streamlines. We support three streamline sources: a point source (tip of the stylus), a line source (Figure 6.52) and a plane source (Figure 6.53).

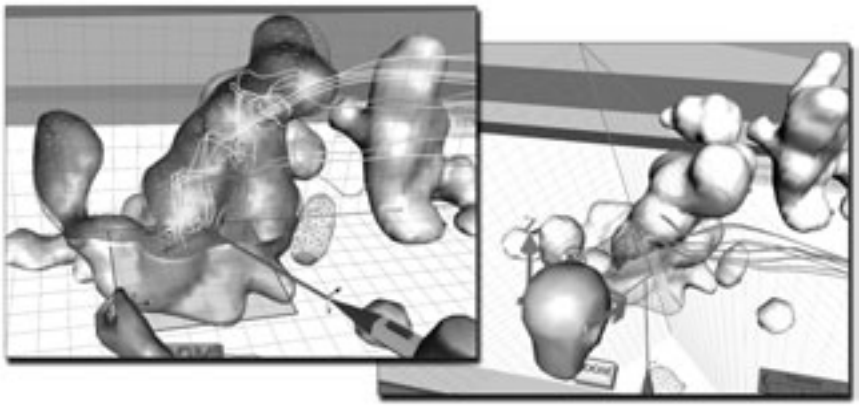


Figure 6.52: Line source of streamlines is attached to the Plexipad.

The line source and plane source, respectively, are attached to the Plexipad. Each time that the Plexipad moves, the streamlines are recomputed from new positions. The dominant hand with the stylus can navigate in the cloud system, operate the 3D GUI or can probe the data.

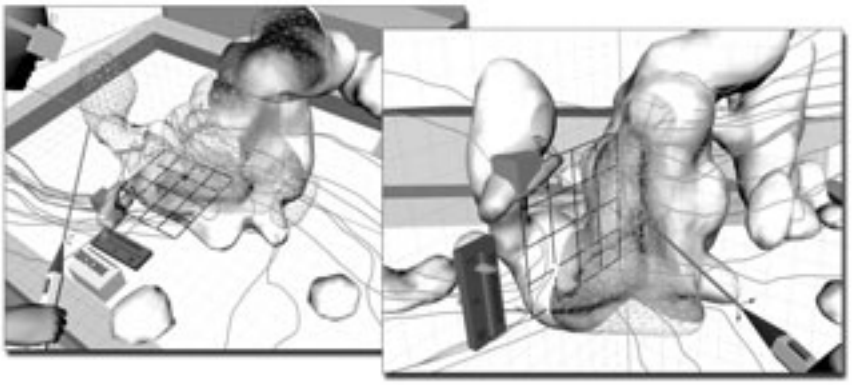


Figure 6.53: Plane source of streamlines is attached to the Plexipad.

In the same fashion the Plexipad can be used as a starting basis for particle tracing. Special attention has to be given to the time dependence of this dataset and a special time-dependent integration scheme has to be used. But we have left this technique for future work.

6.3.7 Cloud Visualization: Summary and Discussion

We have used several different visualization tools and configurations of the visualization pipeline to visualize the various modalities of the cumulus clouds. Although the use of the point-based cloud visualization was good for a fast overview of the cloud data, the pre-processed iso-surfaces provided a much better volumetric visualization. To examine the interior of the clouds, an OpenGL-based clipping-plane scheme has been implemented. The cloud clipping can be naturally used in combination with the direct data slicer, which is attached to the Plexipad, see Figure 6.54.

Figure 6.54: The user performs a two-handed exploration of the cloud data. The non-dominant hand holds the Plexipad with clipping planes and direct data slicer, while the dominant hand navigates and probes the data (see also the Color Section).

The VRX framework handles the parallel processing of the data reading and the update of the visualization tools. Therefore, the frame rate of the system is well above 10 Hz, which results in a comfortable and interactive VE. When playing back the simulation data using the time controller, the latency caused by the hard disk reading and the tool updates is noticed from the slow update of the visualization information. This however does not strongly affect the performance and interactivity of the tools and the VE. The best play back frame rate

can be achieved if only the necessary data files are read into memory (thus reducing harddisk reading time). An advanced data management scheme based on context data cropping is desirable.

Preliminary user tests with dr. Harm Jonker, who has provided us with the simulation data, were very successful. He experienced no difficulties in navigating through the data and was satisfied with the intuitive control of the data slicers with the Plexipad. Using a correctly calibrated texture slicer, which visualized vertical air momentum, he was able to identify the shells of downward air flow alongside the cumulus clouds (Figure 6.54). He was very enthusiastic about the use of the vector data slicer in combination with the cutting plane. Using this combination of tools on the Plexipad, he was able to cut trough a cloud and inspect the momentum inside and outside the clouds, see Figure 6.49. He noticed several interesting turbulent phenomena and unexpected air flow inside the clouds. The intuitive (two-handed) control of the environment and tools allowed him to directly inspect the data without great effort.

Based on these promising results, our research group is planning to continue on this project. This will concentrate on the implementation tuning of automatic extraction, clustering and tracking of clouds and other features such as vortices, or places where turbulence occurs. In addition more visualization techniques (e.g. particle tracing and streamlines in time-dependent datasets) and some exploration tools (e.g. region of interest in data) will be added.

Chapter 7

Conclusions and Future Work

This thesis describes research in the field of interaction and scientific visualization in virtual environments. In this chapter, we summarize the main results and draw overall conclusions. At the end, we give directions for future research.

7.1 Conclusions

Based on our survey (Chapter 2) of related work in the fields of Virtual Reality and scientific visualization we have proposed our research agenda for this thesis, which included the following main research topics: environments for developing VR applications and design of VEs, interaction techniques for VEs, alternative force feedback approaches, techniques and architectures for interactive data visualization and exploration in VEs, and computational steering environments. This research has been performed on the Responsive Workbench. During this research project we developed the RWB Library and RWB Simulator, our basic software environment. On top of this we designed and implemented the VRX toolkit, an interactive visualization toolkit for the Workbench.

Concepts described in this thesis have been applied in several case studies, dealing with visualization of scientific data on the RWB. Within the Molecular Dynamics case study we developed MolDRIVE, a system for visualization and steering of MD simulations. The cumulus clouds case study was a test-bed application during the development of VRX.

7.1.1 Development of Workbench Applications

In Chapter 3, we introduced the Virtual Workbench (equivalent name for the Responsive Workbench) and described the technical characteristics of this system. Section 3.3 discusses the design aspects of applications and VEs for the Workbench. The RWB offers a large screen to visualize 3D models. It intensifies 3D perception of the models and the data, and combined 2D and 3D interfaces can be used for the user interaction. Workbench applications use the laboratory table metaphor or the window-on-the-world metaphor.

Section 3.4 presents basic concepts of the RWB Library. This library provides a solid basis for developing VR applications for the Workbench and adds a necessary "VR functionality and behaviour" to the graphical objects, provided by Performer and OpenGL.

The problem of developing and debugging applications has been addressed with the RWB Simulator, see Section 3.5. The simulator provides the developer

with a birds-eye view on a simulated Workbench which runs the actual application. The ability to play back a recorded Workbench session in the simulator helps the developer during testing, debugging, previewing and creating still-images and animations. Although the idea of a recording and playback option seems simple and straightforward, most VR software environments do not provide this option. The RWB Simulator has proved to be a very useful tool.

7.1.2 Interaction with Virtual Environments

Chapter 4 deals with interaction techniques in virtual environments. Based on analysis of existing interaction techniques for object selection and manipulation, and for navigation in the VE, we presented interaction techniques suitable for the RWB, see Section 4.1. Collision detection, collision handling, and object constraints are important for realistic behaviour during manipulation of objects in VEs. We presented our approach of interactive object collision detection and handling, and a geometric approach to handling object constraints.

Section 4.2 presents a virtual force feedback method for manipulation of virtual objects. We introduced the spring-based manipulation tools (spring, spring-fork and spring-probe), which use the spring metaphor, providing visual force feedback. The spring-fork is designed for direct object manipulation. The deformation of the spring part of the tool (stretch, bend and twist) shows the forces and torques acting on the tool. The spring-tools also give a visual feedback during object collisions. The spring tools were tested on a simple assembly task. Informal user tests indicate that these tools are very easy to use, and provide realistic visual feedback.

The spring-fork has been adapted for distant object manipulation. We have developed the spring manipulator, which has been used for particle steering (manipulation with atoms) in a real-time Molecular Dynamics simulation, visualized on the Workbench using the MolDRIVE system. In Section 4.3, we described three methods for particle steering, of which the virtual spring manipulator, to exert an external force by the user on a manipulated particle, seems the most intuitive steering tool. The steering tools have been tested in electrolyte and protein studies with MolDRIVE. These tools proved very successful. The complete description of MolDRIVE case studies is in Section 6.2.

7.1.3 Exploration and Data Visualization in VEs

Chapter 5 presents our approach of exploration and data visualization in VR, and the VRX toolkit. The description begins with our interaction and exploration tools, see Section 5.1. We defined one- and two-handed interaction scenarios, which have been used for development of the tools. The tool set contains tools for navigation (zoom tool, mini system tool, etc.), selection (ROI), probing of data (point, line, plane and volume probe), and tools for particle steering.

We employed the real-world metaphors such as: magnifying glass (zoom tool), pen and notepad (two-handed use of probing tools with the stylus in the dominant hand and the Plexipad in the non-dominant hand), fishing rod (spring manipulator), and other metaphors.

The two-handed interaction scenario, holding two dimensions of the Plexipad in one hand and controlling the third dimension by the stylus in the other, is useful for designing new easy-to-use interaction tools. This two-handed synergy between the stylus and the Plexipad is very intuitive and allows natural exploration and probing of volumetric datasets.

The described interaction scenarios provide a good approach for development of navigation and probing tools. These tools were initially developed for the MolDRIVE system and were later incorporated in the VRX toolkit.

After presenting the intuitive interaction and exploration tools we described our data visualization framework, the VRX toolkit, see Section 5.2. This toolkit enables rapid development of visualization applications for the Workbench. VRX uses a multiprocessing scheme and adaptive resolution of the probing tools for enhancement of performance. The VRX tools were designed to be practical and easy-to-use for data visualization in a VE. The VRX toolkit has been successfully used for visualization of a time-dependent dataset in the cumulus clouds case study, described in Section 6.3.

7.1.4 Lessons Learned

Development of VR Applications

While VR technology and VR software still urgently need standardization, several open source initiatives try to address this problem (VR Juggler, OpenSceneGraph, etc.). High-end VR visualization applications are usually built in toolkits on top of Performer and OpenGL. In this sense, the decision we made 4 years ago to build our own VR library was right. Due to this, we have also developed a broad knowledge about VR systems and implementation of high performance graphics and visualization software for VR.

Simulator environments are more or less necessary for development of VR applications. With our concept of the RWB Library and the RWB Simulator we go beyond conventional VR libraries, supporting the simulator mode (i.e. the CAVE library and the CAVE simulator). In 1999, we observed that VE session recording and playback facilities were very useful, yet not available in most VR systems. In 2003, this observation is still true. Although this feature is relatively easy to implement in a VR library, most of VR systems simply ignore the possibility to record user interaction with the application in the VE. In our system, this simple feature increases the productivity in developing VR applications, with a lot of user interaction, which has to be carefully designed and tested.

Interaction with VEs

Interaction is the most important issue in today's VR. Although spatial interaction seems very natural, many users have problems in performing spatial interactions in VEs. Providing a good stereo display is not at all enough. The problems with 3D interaction are caused often by (too) inaccurate tracking of input devices, wrong visual feedback during interaction, bad system response due to high latency, not stimulating other sensations (acoustic, haptic or tactile feedback), and sometimes complicated or un-natural usage of interaction tools.

Indeed, technical solutions include improvements in tracking accuracy, minimizing of the system latency by optimizing the application processing pipeline, and integration of additional devices to provide acoustic, haptic (force) or tactile feedback.

In our system, we highlight the selected or manipulated objects and optionally provide sounds during the interaction (touch, un-touch, select, manipulate and release events). This kind of feedback helps enormously. For further improvement we use passive haptic feedback provided by the physical surface of the Plexipad and the screen of the Workbench. We usually put our interaction widgets there. The effect is that the user feels a physical contact with an object of the VE.

Less obvious solutions deal with a better visual feedback, a better design of VEs, a better design of interaction tools with utilization of interaction metaphors from the real world. Virtual Reality for data visualization purposes does not need to mimic reality, but from our experience we can conclude that using real-world metaphors (*visual* and *behaviour metaphors*) helps the users to work in the VEs.

Even a proper visual metaphor can utilize the dominance of human vision to replace other human senses. An example of this is our approach of visual force feedback. We use a *visual spring metaphor* to display forces during manipulation. The spring-tool creates a flexible connection between the physical interaction device and manipulated virtual objects. Further, the spring-tool can be used as an input of the force into the VE.

Informal user tests have shown that most users very quickly accommodate to such visual force feedback, even without exerting a physical force. The users very easily learn to use manipulation techniques that are based on this principle. Our experience is that after a few seconds even VR non-experienced subjects could use the spring-fork for the assembly task. Certainly, most people are skilled at using a physical fork. Thus, we do not need to teach them to use it. But the functionality of the physical fork is completely different from the virtual spring-fork. Even then, when it is a good metaphor, people can efficiently use such a manipulation tool.

The fishing rod metaphor, used in the spring particle manipulator, proved also to be successful. For the users it is usually difficult at the beginning to get used to the ray-casting technique, which is needed for particle selection. After learning to select a particle, it is already easy to manipulate and exert a virtual force on the particle. One important aspect is that the users must get used to another kind of inertial forces, as they interact with very small atomic systems.

A good example of a *behaviour metaphor* for interaction is the pen and notepad metaphor, represented with physical input devices (stylus and Plexipad). In this case, the VE generates content and functionality of the virtual notepad as opposed to the spring-tools, where the VE generates the tool itself. Also this procedural metaphor helps people to use interaction techniques, derived from such a metaphor.

At this point, we can conclude that for designing interaction techniques it is important to look for suitable metaphors from real-life experience.

Data Visualization in VEs

From our research it became obvious that volumetric and multi-dimensional time-evolving data, possibly containing complex 3D structures, can be visualized and explored in a much more natural way in VEs, possibly sooner leading to scientific (re)discoveries in the simulated data.

This experience has been obtained during working on the case studies and co-operating with the scientists from other research groups, whose simulation data we were visualizing on the Workbench. The scientists were rather enthusiastic about our techniques for interactive visualization on the Workbench and about the new possibilities that VR offers. We have also tried to answer real scientific questions from their research domains with the visualization on the Workbench. Visualization in VR can give relatively quickly an answer but we need to know what to look for.

We have learned that visualization in VEs is strongly dependent on the interaction techniques. Therefore we have studied the methods of interaction extensively, see Chapters 4 and 5. Visualization of data in VR is more exploratory in nature than on desktop systems. The visualization environment should thus encourage intuitive exploration of the VE, containing visual representations of the data, and support quick probing, and measurements on the data. The usage of good metaphors for the tools and for the tasks can help users to explore the data. Two-handed data exploration with the Plexipad and the stylus has proven to be an effective approach.

Another aspect is that from VEs an interactive response to user's actions is expected. Visualization, probing and exploration tools for VEs should enhance the responsive nature of the VEs.

The VRX approach forms a solid basis for semi-immersive visualization on the Responsive Workbench. The computational steering and visualization environment in the MolDRIVE system has also clearly demonstrated the advantages of steering remotely running simulations and interactive visualization of the data in VEs, as shown on the Workbench and in the CAVE.

7.2 Future Work

Obviously, future work includes extension of the concepts and the techniques described in this thesis. The work should also try to address several general long-term research topics.

7.2.1 Extension of Concepts and Techniques Developed

The VRX concept should be extended with a generalized computational steering environment to support the "ultimate Workbench metaphor", a visual and interactive VR laboratory. A good computational steering of remote simulations, running on supercomputers, is therefore desirable. The VRX toolkit can be further improved in volumetric visualization techniques for VR, such as direct volume rendering and interactive iso-surface generation.

The research with MolDRIVE and protein modeling should definitely be continued. Improvements in MolDRIVE performance using the VRX toolkit are necessary. Finding better and faster particle representations is desirable, so that we can visualize more than ten thousand particles. The particle steering techniques should be extended for manipulation of groups of particles with new multi-spring tools.

The cloud visualization in the cumulus clouds case study has demonstrated that VRX works well for static frames of the dataset, but several open problems remain: visualization of large time-dependent datasets (i.e. generation of streamlines, or particle tracing); how to play back or animate the large time-evolving datasets in VEs. The most crucial requirement is here to keep frame rates at the high level of interactivity.

To get closer to the end-users of visualization in VR (scientists, non-VR&VIS experts), it should be studied whether integration of VRX with an existing network builder visualization system, such as OpenDX or AVS Express, could help with a better acceptation of this technology for applications in the scientific domain. We believe that extension of the network builder concept might help.

Although we have clearly demonstrated utilization of VR for visualization and exploration of scientific data on several case studies, more evidence is still needed and we need to perform formal usability and validation studies.

7.2.2 Long-Term Topics

The technological VR issues will still include: improvement of display technologies, more realistic rendering, maximizing rendering speed, minimizing the system latency, more accurate and wider range tracking, and better acoustic and haptic augmentation of VEs.

Virtual Reality needs to get closer to the potential users and needs to find more production-stage applications. Although a number of experimental VR toolkits exist, it is still difficult to implement an application for a given VR system. Therefore two things are desirable: better standards, both on the hardware and the software level, and VR must become affordable. One of the current trends, also for scientific visualization, is the development of personal desktop VR systems, driven by relatively cheap PCs and equipped with affordable display technology, instead of the expensive large projection-based systems.

Each VR system has its own specifics, determining possible styles of interaction. There is still a great demand for natural and effective user interfaces and interaction techniques for existing VR systems. For technical and conceptual reasons the techniques that work well in HMDs or in CAVEs do not always work properly (or do not make any sense) on Workbenches, and vice versa. For example, on the Workbench the user does not need to look around or fly through a virtual world as in CAVEs or HMDs. An improved categorization of the interaction styles and techniques for different VR systems is therefore needed.

In the context of scientific visualization in VEs, interaction will for a long time be an important issue and topic for future research. Selection and manipulation of objects, and navigation in VEs (travel specification, way-finding, keeping context) are generally still not satisfactory. Possible improvements include acoustic and haptic enrichment of interaction, more conscious utilization of real-world metaphors, and recognition of interaction gestures. Use of the voice as an additional input device and speech recognition have a potential to improve natural communication between a human and a computer or another human in collaborative VEs.

Visualization tools for VEs are expected to be highly interactive, with a fast response. Due to the large size of data or complexity of visualization techniques used (streamlines, iso-surfaces, volume rendering, etc.), the system response may become very slow. This effectively means that for certain techniques and amounts of data we do not have enough resources yet to guarantee computation and data access in real time.

Part of the solution may be the time-critical computing [Bryson & Johan, 1996; Funkhouser & Séquin, 1993], which should guarantee some result within a given time budget while meeting the user requirements. First, the time budget for various techniques has to be determined. And after that, it has to be decided which visualization algorithm with which parameters meets the time budget.

Accuracy versus speed tradeoffs can be used to keep the computation time of the visualization algorithm within the budget.

In some cases it also seems reasonable to use a hierarchical representation of the data with different levels of detail. During visualization and exploration in the VE it should be possible to switch between the data representations.

Another approach may utilize distributed and parallel visualization. The idea is to run the visualization algorithm on a parallel supercomputer, collect the results and send them to the graphics server that controls the virtual environment. The system and network latencies are here the big issues.

A problem related to potentially very large datasets is that most interaction (selection, manipulation) and visualization tools are based on "small problems". Therefore an effort should be also put in the development of scalable interaction tools (i.e. selection and manipulation of 50 particles from 10.000 in total), and the development of scalable visualization tools for large datasets.

A promising approach to deal with large time-dependent datasets is feature extraction, feature tracking, event detection and iconic visualization [Sadarjoen, 1999; Reinders, 2001]. Instead of working with the large dataset during the exploration phase in the VE, the data should be preprocessed off-line: selection and extraction of features, calculation of the feature attributes, the features can be tracked in time, and finally visualized with icons (spheres, ellipsoids, trees, etc.). Iconic visualization (simple geometrical objects) seems to be very suitable for virtual environments. These feature-processing techniques should be definitely further extended and applied also in the context of data visualization in VEs. The question is, whether the preprocessing phase (selection of a feature, extraction and tracking) could be also performed directly in the immersive VE, or would it still be better to do this step off-line on a desktop system.

A great field for future work is also collaborative visualization in virtual environments. VEs naturally encourage co-operation of scientists in visualization and exploration of the data. In recent years, the phrase "telecollaboration" has been established, when multiple participants interact with a shared dataset and with each other over a network [Sawant *et al.*, 2000].

Finally, we conclude with a question towards the future: *"Will scientists ever use virtual reality and scientific visualization without programmers or service providers to help them?"*. Unfortunately, the VR technology and visualization systems for VR are still very premature and rather difficult to use and to operate for non-experts. It is important that we will continue in convincing the scientists from other research disciplines of the advantages of visualization in VR. It has taken computer graphics several decades to win the trust of the scientific community. Let's hope that VR will be accepted more quickly.

Bibliography

Agrawala, M., Beers, A., Fröhlich, B., Hanrahan, P., McDowall, I., & Bolas, M. 1997. The Two-User Responsive Workbench: Support for Collaboration Through Individual Views of a Shared Space. *Pages 327–332 of: Proc. ACM SIGGRAPH '97*.

Arthur, K., Preston, T., Taylor II, R.M., Brooks, F.P., Whitton, M.C., & Wright, W.V. 1998. Design and Building the PIT: a Head-Tracked Stereo Workspace for Two Users. *In: Proc. Immersive Projection Technology '98*.

Barzel, R., & Barr, A.H. 1988. A Modeling System Based on Dynamic Constraints. *ACM Computer Graphics*, **22**(4), 179–188.

Beckers, J. 1999. *Molecular Simulations of porous silica and β-Alumina*. Ph.D. thesis, TU Delft, Department of Applied Sciences, Computational Physics Section.

van den Bergen, G. 1999. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects. *Journal of Graphics Tools*, **4**(2), 7–25. `http://www.win.tue.nl/~gino/solid/`.

Bier, E.A., Stone, M.C., Pier, K., Buxton, W., & DeRose, T. 1993. Toolglass and Magic Lenses: The see-through interface. *Pages 73–80 of: Proc. ACM SIGGRAPH '93*.

Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., & Cruz-Neira, C. 2001. VR Juggler: A Virtual Platform for Virtual Reality Application Development. *Pages 89–96 of: Proc. IEEE Virtual Reality '01,*.

Bowman, D., & Hodges, L. 1995. *User Interface Constraints for Immersive Virtual Environment Applications*. Tech. rept. GIT-GVU-95-26. Georgia Institute of Technology. Technical Report.

Bowman, D., & Hodges, L. 1997a. An Evaluation of Techniques for Grabbing and Manipulation Remote Objects in Immersive Virtual Environments. *Pages 35–38 of: Proc. ACM Interactive 3D Graphics '97*.

Bowman, D., & Hodges, L. 1997b. User Interface Constraints for Immersive Virtual Environment Applications. *Pages 35–38 of: Proc. IEEE VRAIS '97*.

Brederson, J.D., Ikits, M., Johnson, C.R., & Hansen, C.D. 2000. The Visual Haptic Workbench. *Pages 46–49 of: Proc. Phantom Users Group Workshop '00*.

Brinckman, M. 2002. *Visualization of Cumulus Clouds on the Virtual Workbench*. Tech. rept. DUT-ITS-CG-02-07. TU Delft, Department of Information Technology and Systems, Computer Graphics Section. Internal research report (in Dutch).

Brooks, F.P. 1988. Grasping reality through illusion - interactive graphics serving science. *Pages 1–11 of: Proc. ACM Human Factors in Computer Systems '88*.

Brooks, F.P. 1999. What's Real About Virtual Reality. *IEEE Computer Graphics and Applications*, **19**(6), 16–27.

Brown, J.M., & Colgate, J.E. 1994. Physics-based Approach to Haptic's Display. *In: Proc. ISMCR '94*. Topical Workshop on Virtual Reality.

de Bruin, P.W., Vos, F.M., Post, F.H., Frisken-Gibson, S.F., & Vossepoel, A.M. 2000. Improving Triangle Mesh Quality with SurfaceNets. *Pages 804–813 of: Medical Image Computing and Computer-Assisted Intervention – Proc. MICCAI '00*. Pittsburgh, PA, USA.

Bryson, S. 1994a. Approaches to the Successful Design and Implementation of VR Applications. *In: Course Notes, ACM SIGGRAPH'94*.

Bryson, S. 1994b. Virtual Environments in Scientific Visualization. *Pages 201–220 of: Proc. ACM VRST'94*.

Bryson, S., & Johan, S. 1996. Time Management, Simultaneity, and Time-critical Computation in Interactive Unsteady Visualization Environments. *Pages 255–261 of: Proc. IEEE Visualization '96*.

Bryson, S., & Levit, C. 1992. The Virtual Windtunnel. *IEEE Computer Graphics and Applications*, **12**(4), 25–34.

Bues, M., Blach, R., Stegmaier, S., Häfner, U., Hoffman, H., & Haselberger, F. 2001. Towards a Scalable High Performance Application Platform for Immersive Virtual Environments. *Pages 165–174 of: Proc. Immersive Projection Technology and Eurographics Virtual Environments '01*.

Burdea, G.C. 1996. *Force and Touch Feedback for Virtual Reality*. Publishers City. ISBN 0-471-02141-5.

Burdea, G.C., & Coiffet, P. 1994. *Virtual Reality Technology*. Jonh Wiley.

Cohen, J., Lin, M., Manocha, D., & Ponamgi, K. 1995. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments. *Pages 189–196 of: Proc. ACM Interactive 3D Graphics '95*. `http://www.cs.unc.edu/~geom/I_COLLIDE.html`.

Coquillart, S., & Wesche, G. 1999. The Virtual Palette and the Virtual Remote Control Panel: A Device and an Interaction Paradigm for the Responsive Workbench. *Pages 213–216 of: Proc. IEEE Virtual Reality '99*.

Cruz-Neira, C., Sandin, T.A., & de Fanti, R.V. 1993. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. *Pages 135–142 of: Proc. ACM SIGGRAPH '93*.

Cutler, L.D., Fröhlich, B., & Hanrahan, P. 1997. Two-handed Direct Manipulation on the Responsive Workbench. *Pages 107–114 of: Proc. ACM Interactive 3D Graphics '97*.

Dai, P., Eckel, G., Göbel, M., & Wesche, G. 1997. *Virtual Space: VR Projection System Technologies and Applications*. Tech. rept. GMD/IMK. Internal report on AVOCADO framework.

van Dam, A., Forsberg, A.S., Laidlaw, D.H., LaViola, J., & Simpson, R.M. 2000. Immersive VR for Scientific Visualization: A Progress Report. *IEEE Computer Graphics and Applications*, Nov/Dec, 26–52.

Darken, R., & Sibert, J.L. 1993. A Toolset for Navigation in Virtual Environments. *Pages 157–165 of: Proc. ACM SIGGRAPH '93*.

Durbeck, L.J.K., Macias, N.J., Weinstein, D.M., Johnson, C.R., & Hollerbach, J.M. 1998. SCIrun Haptic Display for Scientific Visualization. *In: Proc. Phantom Users Group Meeting '98*.

Durbin, J., SwanII, J.E., Colbert, B., Crowe, J., King, R., King, T., Scannell, Ch., Wartell, Z., & Welsh, T. 1998. Battlefield Visualization on the Responsive Workbench. *Pages 463–466 of: Proc. IEEE Visualization '98*.

Durlach, N.I., & Mavor, A.S. 1995. *Virtual Reality - Scientific and Technological Challenges*. National Research Council.

Eckel, G., Hiatt, S., & Galgani, D. 1997. *Iris Performer: Programmer's Guide*. Silicon Graphics, Inc. Electronically available from: `http://www.sgi.com/software/performer/manuals.html`.

Foley, J.D., van Dam, A., Feiner, S, & Hughes, J. 1990. *Computer Graphics: Principles and Practice*. Fifth, revised edn. Addison-Wesley Publishing Company.

Foulser, D. 1995. IRIS Explorer: A Framework for Investigation. *IEEE Computer Graphics*, **29**(2), 13–16. `http://www.nag.co.uk/Welcome_IEC.html`.

Fröhlich, B., Tramberend, H., Agrawala, M., & Baraff, D. 2000. Physically-Based Manipulation on the Responsive Workbench. *Pages 5–12 of: Proc. IEEE VR '00*.

Fuhrman, A., Schmalstieg, D., & Gervautz, M. 1998. Strolling Through Cyberspace With Your Hands in Your Pockets: Head Directed Navigation in Virtual Environments. *Pages 216–225 of: Proc. Eurographics Virtual Environments '00*.

Funkhouser, T., & Séquin, C. 1993. Adaptive Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environments. *Pages 247–254 of: Proc. ACM SIGGRAPH '93*.

Garland, M. 1999. *Quadric-Based Polygonal Surface Simplification*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University,Pittsburgh. Software and thesis available at:
`http://www.cs.cmu.edu/~garland/quadrics/`.

Garvic, I. 2001. *Real-time, interactive simulations on the Virtual Workbench*. M.Sc. thesis, TU Delft, Department of Applied Sciences, Computational Physics Section.

Germans, D., Spoelder, H.J.W., Renambot, L., & Bal, H.E. 2001. VIRPI: A High-Level Toolkit for Interactive Scientific Visualization in VR. *Pages 109–120 of: Proc. Immersive Projection Technology and Eurographics Virtual Environments '01*.

Gottschalk, S., Lin, M., & Manocha, D. 1996. OBB-Tree: A Hierarchical Structure for Rapid Interface Detection. *Pages 171–180 of: Proc. ACM SIGGRAPH '96*. `http://www.cs.unc.edu/~geom/OBB/OBBT.html`.

Grant, B., Helser, A., & Taylor II, R.M. 1998. Adding force to a Stereoscopic Head-Tracked Projection Display. *Pages 81–88 of: Proc. IEEE VRAIS '98*.

Guiard, Y. 1987. Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as Model. *Journal of Motor Behaviour*, **19**(4), 486–517.

de Haan, G. 2002. *Interactive Visualization on the Virtual Reality Responsive Workbench*. M.Sc. thesis, TU Delft, Department of Information Technology and Systems, Computer Graphics Section.

de Haan, G., Koutek, M., & Post, F.H. 2002. Towards Intuitive Exploration Tools for Data Visualization in VR. *Pages 105–112 of: Proc. ACM VRST '02*.

Haase, H. 1994. How Scientific Visualization Can Benefit from Virtual Environments. *CWI Quarterly*, **7**(2), 159–174.

Haase, H., Strassner, J., & Dai, F. 1996. VR Techniques for the Investigation of Molecular Data. *Computers and Graphics*, **20**(2), 207–217.

Haber, R.B., & McNabb, D.A. 1990. Visualizations idioms: A conceptual model for scientific visualization systems. *Pages 75–83 of:* Nielson, G.M., Shriver, B.D., & Rosenblum, L. (eds), *Visualization in scientific computing*. IEEE Computer Society Press.

Halle, M. 1997. Autostereoscopic displays and computer graphics. *Computer Graphics*, **31**(2), 58–62.

van Hees, J., & den Hertog, J. 2002. *MolDRIVE: a system for remote interactive MD simulations on a Virtual Reality Responsive Workbench*. M.Sc. thesis, TU Delft, Department of Applied Sciences, Computational Physics Section. `http://visualization.tudelft.nl/~michal/MolDRIVE`.

Herndon, K., van Dam, A., & Gleicher, M. 1994. The challenges of 3D interaction. *SIGCHI Bulletin*, **26**(4), 36–43.

Hibbard, W., & Santek, D. 1990. The VIS-5D System for Easy Interactive Visualization. *Pages 129–134 of: Proc. IEEE Visualization '90*. Available from: `http://www.ssec.wisc.edu/~billh/vis5d.html`.

Hibbard, W., Anderson, J., Foster, I., Paul, B., Jacob, R., Schafer, C., & Tyree, M. 1996. Exploring coupled atmosphere-ocean models using Vis5D. *Int. Journal of Supercomputer Applications*, **10**(2), 211–222.

Hinckley, K., Pausch, R., Goble, J.C., & Kassel, N.F. 1994. Passive real-world interface props for neurosurgical visualization. *Pages 452–458 of: Proc. ACM CHI'94*.

Hubbold, R., Cook, J., Keates, M., Gibson, S., Howard, T., Murta, A., West, A., & S.Pettifer. 1999. GNU/MAVERIK: A micro-kernel for large-scale virtual environments. *Pages 66–73 of: Proc. ACM VRST '99,*.

Humphrey, W., Dalke, A., & K.Shulten. 1996. VMD - Visual Molecular Dynamics. *Journal of Molecular Graphics*, **13**(1), 33–38.

Insko, B.E. 2001. *Passive Haptics Enhances Virtual Environments*. Ph.D. thesis, UNC Chapel Hill, Computer Science Department.

Johnson, A., & Leigh, J. 2001. Tele-Immersive Collaboration in the CAVE Research Network (chapter). *Collaborative Virtual Environments: Digital Places and Spaces for Interaction*, 225–243.

Kabbash, P., Buxton, W., & Sellen, A. 1994. Two-handed input in a compound task. *Pages 444–451 of: Proc. ACM CHI '94*.

Kapoor, A., Leigh, J., Wheless, G., Lascara, C., Johnson, A.E., Park, K.S., & Defanti, T.A. 2000. Cave6D: A Tool for Collaborative, Interactive Immersive Visualization of Environmental Data. *In: Proc. ACM CVE '00*. Available from: `http://www.evl.uic.edu/akapoor/cave6d/`.

Kindratenko, V., & Bennett, A. 2000. Evaluation of Rotation Correction Techniques for Electromagnetic Position Tracking Systems. *Pages 13–22 of: Proc. Eurographics Virtual Environments '00*.

Koutek, M., & Post, F.H. 2000. Dynamics in Interaction on the Responsive Workbench. *Pages 43–54 of: Proc. Eurographics Virtual Environments '00*.

Koutek, M., & Post, F.H. 2001a. A Software Environment for the Responsive Workbench. *Pages 428–435 of:* Lagendijk, R.L., & Heijnsdijk, J.W.J. (eds), *Proc. ASCI '01*. ASCI, Netherlands.

Koutek, M., & Post, F.H. 2001b. Dynamics Manipulation Tools for the Responsive Workbench. *Pages 167–168 of:* Hirose, M., & Tamura, H. (eds), *Proc. International Symposium on Mixed Reality '01*. University of Tokyo, Japan.

Koutek, M., & Post, F.H. 2001c. Spring-Based Manipulation Tools for Virtual Environments. *Pages 61–70 of: Proc. Immersive Projection Technology and Eurographics Virtual Environments '01*.

Koutek, M., & Post, F.H. 2002. The Responsive Workbench Simulator: A Tool for Application Development and Analysis. *Pages 255–262 of:* Skala, V. (ed), *Journal of WSCG '02*.

Koutek, M., van Hees, J., Post, F.H., & Bakker, A.F. 2002. Virtual Spring Manipulators for the Particle Steering in Molecular Dynamics on the Responsive Workbench. *Pages 55–62 of: Proc. Eurographics Virtual Environments '02*.

Krüger, W., Fröhlich, B., Bohn, C.A., Schüth, H., Strauss, W., & Wesche, G. 1995. The Responsive Workbench: A Virtual Work Environment. *IEEE Computer*, July, 42–48.

Kuo, E., Lanzagorta, M., Rosenberg, R., S., Julier, & J., Summers. 1999. VR Scientific Visualization in the GROTTO. *Page 81 of: Proc. IEEE Virtual Reality '99*.

LaViola, J. 2000. MSVT: A Virtual Reality-Based Multimodal Scientific Visualization Tool. *Pages 1–7 of: Proc. IASTED International Conference on Computer Graphics and Imaging '00*.

Lécuyer, A., Coquillart, S., & Kheddar, A. 2000. Pseudo-Haptics Feedback: Can Isometric Input Devices Simulate Force Feedback? *Pages 83–90 of: Proc. IEEE VR '00*.

Leech, J., Prins, J.F., & Hermans, J. 1996. SMD: Visual Steering of Molecular Dynamics for Protein Design. *IEEE Computational Science and Engineering*, **7**(4), 38–45.

Lin, M., & Gottschalk, S. 1998. Collision Detection between Geometric Models: A Survey. *Pages 37–56 of: Proc. IMA Conference on Mathematics of Surfaces '98*.

Lindeman, R., Sibert, J., & Hahn, J. 1999. Hand-Held Windows: Towards Effective 2D Interaction in Immersive Virtual Environments. *Pages 205–212 of: Proc. IEEE Virtual Reality '99*.

Lovejoy, S., Desaulniers-Soucy, N., Lilley, M., & Schertzer., D. 2000. Empirical Analysis of the continuum Limit in Rain. *Pages 402–404 of: Proc. of 13th International Conference on Clouds and Precipitation*.

Lucas, B., Abram, G. D., Collins, N. S., Epstein, D. A., Gresh, D. L., & McAuliffe, K. P. 1992. An Architecture for a Scientific Visualization System. *Pages 107–113 of: Proc. IEEE Visualization '92*.
Formerly IBM Visualization Data Explorer, now open source:
`http://www.opendx.org`.

Massie, T.M., & Salisbury, J.K. 1994. The phantom haptic interface: A device for probing virtual objects. *Pages 295–301 of: Proc. ASME Haptic Interfaces for Virtual Environments and Teleoprator Systems '94*.

McCormick, B.H. 1987. Visualization in scientific computing. *Computer Graphics*, **21**(6). Special Issue.

Meissner, M., Huang, J., Bartz, D., Mueller, K., & Crawfis, R. 2000. A Practical Evaluation of Popular Volume Rendering Algorithms. *Pages 81–90 of: Proc. Volume Visualization and Graphics Symposium '00*.

Mine, M.R. 1995. *Virtual Environment Interaction Techniques*. Tech. rept. TR95-018. UNC Chapel Hill, Computer Science Department.

Mine, M.R. 1996. *Working in a Virtual World: Interaction Techniques Used in the Chapel Hill Immersive Modeling Program*. Tech. rept. TR96-029. UNC Chapel Hill, Computer Science Department.

Mine, M.R. 1998. Making virtual worlds work in a real world. *Pages 38–44 of: Proc. Eurographics Virtual Environments '98*.

Mine, M.R., Brooks, F.P., & Sequin, C.H. 1997. Moving Object in Space: Exploiting Proprioception in Virtual Environments. *Pages 19–26 of: Proc. ACM SIGGRAPH '97*. ACM.

Mirtich, B. 1998. V-Clip: Fast and Robust Polyhedral Collision Detection. *ACM Transactions on Graphics*, **17**(3), 177–208.
`http://www.merl.com/projects/vclip/`.

Moore, M., & Wilhelms, J. 1988. Collision Detection and Responce for Computer Animation. *ACM Computer Graphics*, **22**(4), 289–298.

Mulder, J. D. 1998. Remote Object Translation Methods for Immersive Virtual Environments. *Pages 80–89 of: Proc. Eurographics Virtual Environments '98*.

Mulder, J.D., & van Liere, R. 2002. Personal Space Station. *Pages 73–81 of: Proc. VRIC '02*. Laval Virtual.

Mulder, J.D., van Wijk, J.J., & van Liere, R. 1999. A survey of computational steering environments. *Future Generation Computer Systems*, **15**, 119–129.

Okoshi, T. 1976. *Three-Dimensional Imaging Techniques*. Academic Press.

van de Pol, R., Ribarsky, W., Hodges, L., & Post, F.H. 1999. Interaction Techniques on the Virtual Workbench. *Pages 157–168 of: Proc. Eurographics Virtual Environments '99*.

Poston, T., & Serra, L. 1996. Dextrous virtual work. *CACM*, **39**(5), 37–45.

Poupyrev, I., Weghorst, S., Billinghurst, M., & Ichikawa, T. 1996. The Go-Go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR. *Pages 79–80 of: Proc. ACM UIST '96*.

Poupyrev, I., Weghorst, S., Billinghurst, M., & Ichikawa, T. 1997. A Framework and Testbed for Studying Manipulation Techniques for Immersive VR. *Pages 21–28 of: Proc. ACM VRST '97*.

Prins, J.F., Hermans, J., Mann, G., Nyland, L.S., & Simons, M. 1999. A Virtual Environment for Steered Molecular Dynamics. *Future Generation Computer Systems*, **15**.

Rantzau, D., Frank, K., Lang, U., Rainer, D., & Wössner, U. 1998. COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data. *In: Proc. Immersive Projection Technology '98*.
`http://www.hlrs.de/organization/vis/covise/`.

Raskar R. et al. 1998. The Office of the Future: A Unified Approach to Image-based Modelling and Spatially Immersive Displays. *Pages 179–188 of: ACM SIGGRAPH '98, Course Notes*.

Razzaque, S., Swapp, D., Slater, M., Whitton, M.C., & Steed, A. 2002. Redirected Walking in Place. *Pages 123–129 of: Proc. Eurographics Virtual Environments '02*.

Reinders, F. 2001. *Feature-Based Visualization of Time-Dependent Data*. Ph.D. thesis, TU Delft, Department of Information Technology and Systems, Computer Graphics Section.

Rohlf, J., & Helman, J. 1994. Iris Performer: A High Performance Multiprocessor Toolkit for Realtime 3D Graphics. *Pages 381–394 of: Proc. ACM SIGGRAPH '94*.
Formerly Iris Perfomer, now OpenGL Performer:
`http://www.sgi.com/software/performer/`.

Ruspini, D.C., Kolarov, K., & Hatib, O. 1997. The haptic display of complex objects. *Pages 345–352 of: Proc. ACM SIGGRAPH '97*.

Sadarjoen, A. 1999. *Extraction and Visualization of Geometries in Fluid Flow Fields*. Ph.D. thesis, TU Delft, Department of Information Technology and Systems, Computer Graphics Section.

Sawant N. et al. 2000. The Tele-Immersive Data Explorer (TIDE): A Distributed Architecture for Collaborative Interactive Visualization of Large Data Sets. *In: Proc. Immersive Projection Technology '00,*.

Sayle, R.A., & Milner-White, E.J. 1995. RASMOL: Biomolecular Graphics for All. *Trends in Biochemical Sciences*, **20**(9), 374.
`http://www.umass.edu/microbio/rasmol`.

Schmalstieg, D., Fuhrmann, A., Szalavari, Z., & Gervautz, M. 1998. "Studierstube" - An Environment for Collaboration in Augmented Reality. *Virtual Reality - Systems, Development and Applications*, **3**(1), 37–49.

Schmalstieg, D., Encarnacao, L.M., & Szalavari, Z. 1999. Using Transparent Props for Interaction With The Virtual Table. *Pages 147–154 of: Proc. ACM Symp. Interactive 3D Graphics '99*.

Schroeder, W., Martin, K., & B.Lorensen. 1999. *The Visualization Toolkit*. 2nd edn. Prentice Hall PTR. `http://public.kitware.com/VTK`.

Serra, L., Hern, N., Choon, C.B., & Poston, T. 1997. Interactive Vessel Tracing in Volume Data. *Pages 131–137 of: Proc. ACM Interactive 3D Graphics '97*.

Serra L. et al. 1999. An interface for precise and comfortable 3D work with volumetric medical datasets. *Pages 329–334 of: Proc. Medicine Meets Virtual Reality: 7*.

Siebesma, A.P., & Jonker, H.J.J. 2000. Anomalous scaling of cumulus cloud boundaries. *Phys. Rev. Letters*, **85**(1), 214–217.

Slater, M., Usoh, M., & Steed, A. 1995. Taking Steps: The Influence of a Walking Technique on Presence in VR. *Transactions on Computer-Human Iteraction*, **2**(3), 201–219.

Smith, G., & Stürzlinger, W. 2001. On the Utility of Semantic Constrains. *Pages 41–50 of: Proc. Immersive Projection Technology and Eurographics Virtual Environments '01*.

Smychliaev, I. 1998. *Building a VR visualization application with a modular visualization environment*. Tech. rept. DUT-ITS-CG-98-08. TU Delft, Department of Information Technology and Systems, Computer Graphics Section. Internal research report.

Stoev, S.L., Schmalstieg, D., & Strassen, W. 2001. Two-Handed Through-the-Lens Technique for Navigation in VEs. *Pages 51–60 of: Proc. Immersive Projection Technology and Eurographics Virtual Environments '01*.

Stone, J.E., Gullingsrud, J., & Schulten, K. 2001. A System for Interactive Molecular Dynamics Simulation. *Pages 191–194 of: Proc. ACM Symposium Interactive 3D Graphics '01*.

Stull, R.B. 1988. *An Introduction to Boundary Layer Meteorology*. Revised edition 1997 edn. Kluwer Academic Publishers. ISBN 90-277-2768-6.

Sutherland, I.E. 1970. Computer Displays. *Scientific American*, **222**(June), 57–81.

Swan, J.E., Mueller, K., Moeller, T., Shareef, N., Crawfis, R., & Yagel, R. 1997. An anti-aliasing technique for splatting. *Pages 197–204 of: Proc. IEEE Visualization '1997*.

Szalavari, Z., & Gervautz, M. 1997. The Personal Interaction Panel - a Two-Handed Interface for Augmented Reality. *Eurographics '97, Computer Graphics Forum*, **16**(3), 335–346.

Taubin, G. 2000. Geometric Signal Processing on Polygonal Meshes. *In: State of the Art Report, Eurographics 2000*.
`http://www.research.ibm.com/people/t/taubin`.

Taylor II, R.M. 1999. *Scientific Applications of Force Feedback: Molecular Simulation and Microscope Control*. SIGGRAPH '99, Course notes.

Tramberend, H. 1999. AVANGO: A Distributed Virtual Reality Framework. *Pages 14–21 of: Proc. IEEE Virtual Reality '99*. `http://imk.gmd.de/`.

Upson, C., Faulhaber, T, Kamins, D., Laidlaw, D., Schleigel, D., Vroom, J., Gurwitz, R., & van Dam, A. 1989. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications*, July, 30–42. `http://www.avs.com`.

Wernecke, J. 1994. *The Inventor Mentor*. Reading, Massachusetts, U.S.A.: Addison-Wesley.
Formerly SGI Inventor, now Open Inventor:
`http://www.sgi.com/software/inventor/manuals.html`.

Wheless, G.H., Lascara, C.M., Cox, D., Patterson, R., Levy, S., & Hibbard, W. 1998. Cave5D and Virtual Director: Collaborative visualization and management of large, multivariate environmental datasets. *In: Alliance '98*. Available from: `http://www.ccpo.odu.edu/~cave5d/homepage.html` New version: `http://www-unix.mcs.anl.gov/~mickelso/CAVE2.0.html`.

von Wiegand, T., Schloerd, D., & Sachtler, W. 1999. Virtual Workbench: Near Field Virtual Environment System with Applications. *Presence*, **8**(5), 492–519.

Yoshida, S., Yamada, K., Mochizuki, K., Aizawa, K., & Saito, T. 2002. Scope-based Interaction: A Technique for Interaction in a Image-based VE. *Pages 139–147 of: Proc. Eurographics Virtual Environments '02*.

Zachmann, G. 2000. *Virtual Reality in Assembly Simulation - Collision Detection, Simulation Algorithms, and Interaction Techniques*. Ph.D. thesis, Darmstadt University of Technology.

Internet References:

Web-AVS-int. *The International AVS Centre (IAC)*.
`http://www.iavsc.org/`.

Web-Barco. *Barco projection systems*.
`http://www.barco.com/projection_systems`.

Web-CAVE5D. *Cave5D atmospheric visualization package*.
`http://www.ccpo.odu.edu/~cave5d/`.

Web-CAVElib. *The CAVE Library and the CAVE Simulator*.
`http://www.ncsa.uiuc.edu/VR/`.

Web-DelftHydraulics. *WL|Delft Hydraulics*.
`http://www.wldelft.nl/`.

Web-Dextro. *Volume Interactions*.
`http://www.volumeinteractions.com`.

Web-Dresden3D. *SeeReal Technologies*.
`http://www.seereal.com`.

Web-FakespaceSystems. *FakeSpace Systems*.
`http://www.fakespacesystems.com`.

Web-Fastrak. *Fastrak tracking systems*.
`http://www.polhemus.com/ftrakds.htm`.

Web-MD-demmpsi. *DEMMPSI - MD simulation software*.
`http://www.cp.tn.tudelft.nl/FAQ/MD/Main.html`.

Web-MD-gromacs. *GROMACS - MD simulation software*.
`http://www.gromacs.org`.

Web-MD-namd. *NAMD - MD simulation software*.
`http://www.ks.uiuc.edu/Research/namd/namd.html`.

Web-MD-particle-steering. *MolDRIVE and particle steering in MD*.
`http://visualization.tudelft.nl/~michal/MolDRIVE`.

Web-MR-toolkit. *Minimal Reality (MR) Toolkit*.
`http://web.cs.ualberta.ca/graphics/MRToolkit.html`.

Web-OpenGL. *OpenGL API (Application Programming Interface)*.
`http://www.sgi.com/software/opengl`.

Web-OpenScenegraph. *OpenSceneGraph : cross-platform (C++/OpenGL) library for real-time visualization*. `http://www.openscenegraph.org/`.

Web-Reachin. *Reachin Systems*.
`http://www.reachin.se`.

Web-RWB-lib. *RWB-Library and Simulator*.
`http://visualization.tudelft.nl/~michal/RWBlib`.

Web-SGI-Onyx. *SGI Onyx family*. `http://www.sgi.com/visualization/onyx/`.

Web-SpringMass. *Spring-Mass Simulator*.
`http://links.math.rpi.edu/devmodules/mechanicalosc/springmass`.

Web-SpringTools. *Spring manipulation tools for VR*.
`http://visualization.tudelft.nl/~michal/SpringTools`.

Web-Stereographics. CrystalEyes shutter glasses and other VR hardware:
`http://www.stereographics.com/`.

Web-SVE-lib. *The Simple Virtual Environment (SVE) Library*.
`http://www.cc.gatech.edu/gvu/virtual/SVE/`.

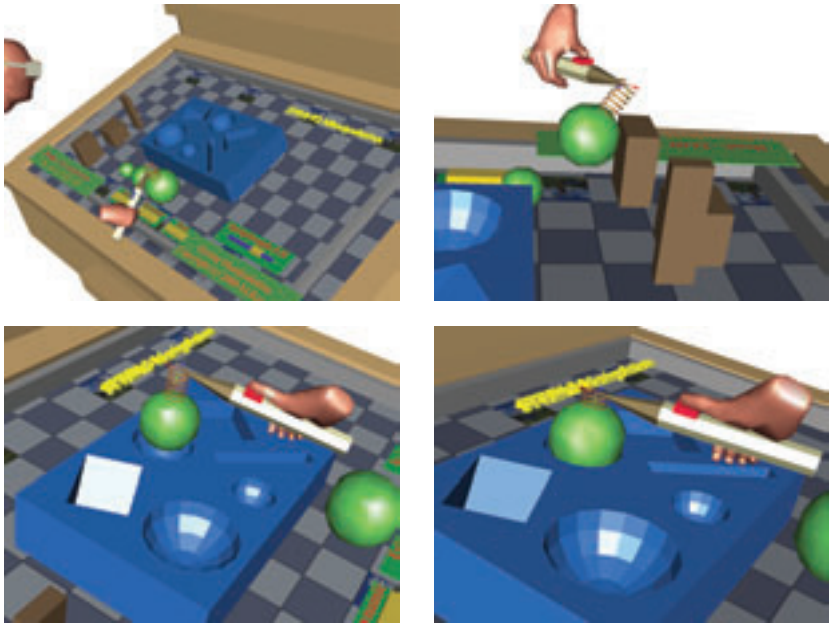
Web-TAN. *TAN systems*. `http://www.tan.de`.

Web-VR-lib. *VRlib, Grotto-viewer for AVS*.
`http://www.ait.nrl.navy.mil/people/ekuo/vrlib-doc`.

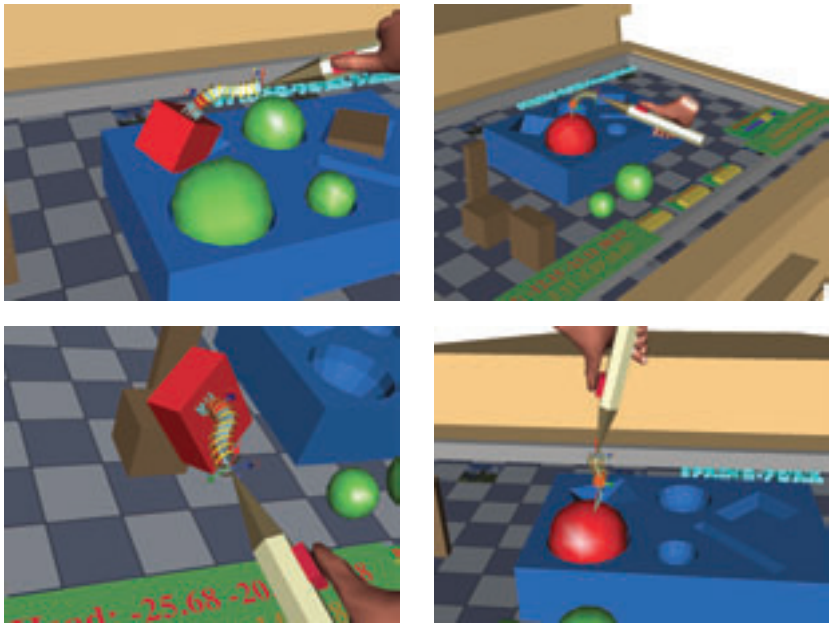
Web-VR-toolkit. *VR Developers Toolkit*.
`http://www.lincom-asg.com/VrTool/`.

Web-VTK-to-Perf. Conversion of VTK geometry into Performer:
`http://brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF/`.

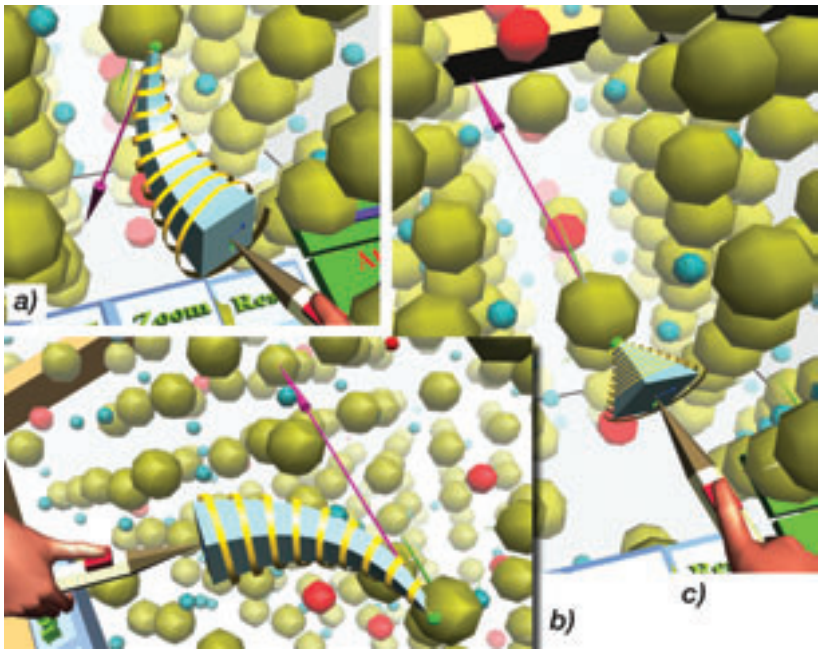
Color Section



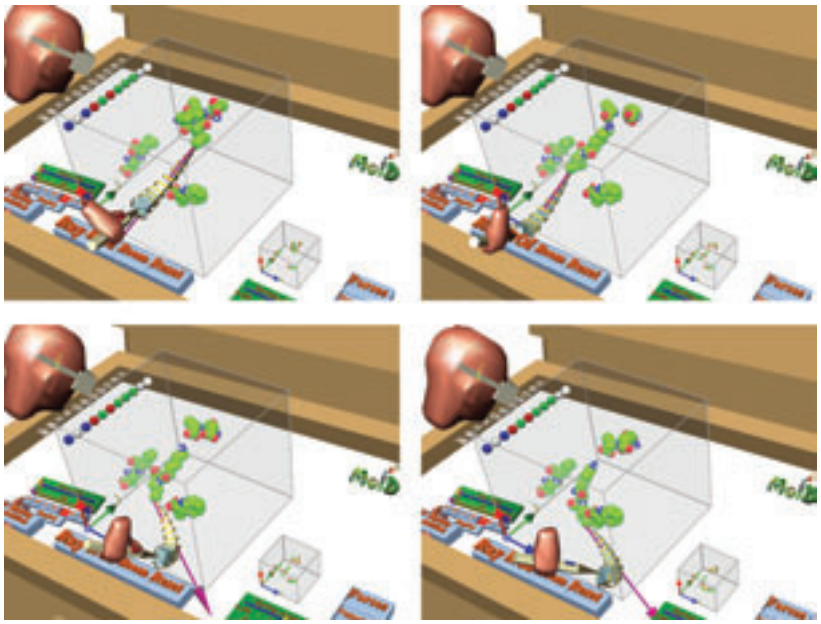
Color Figure 1: Spring tool demonstration (see also Fig. 4.22 on p. 90)



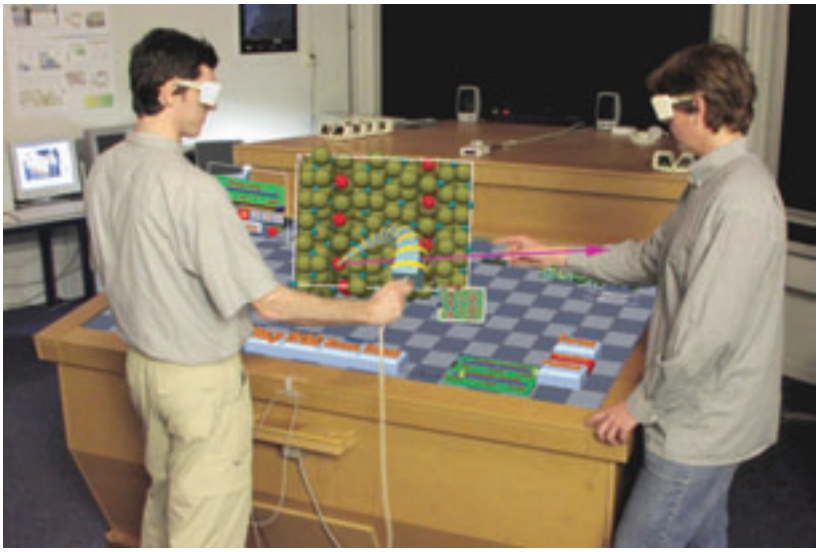
Color Figure 2: Spring-fork demonstration (see also Fig. 4.31 on p. 97)



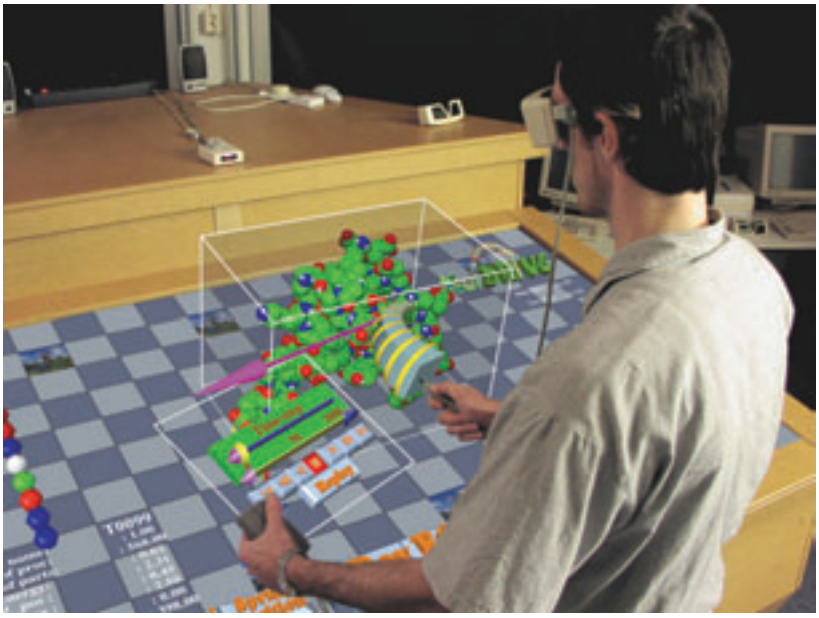
Color Figure 3: MolDRIVE (DEMMPSI): particle steering performed with the spring manipulator (see also Fig. 4.46 on p. 114)



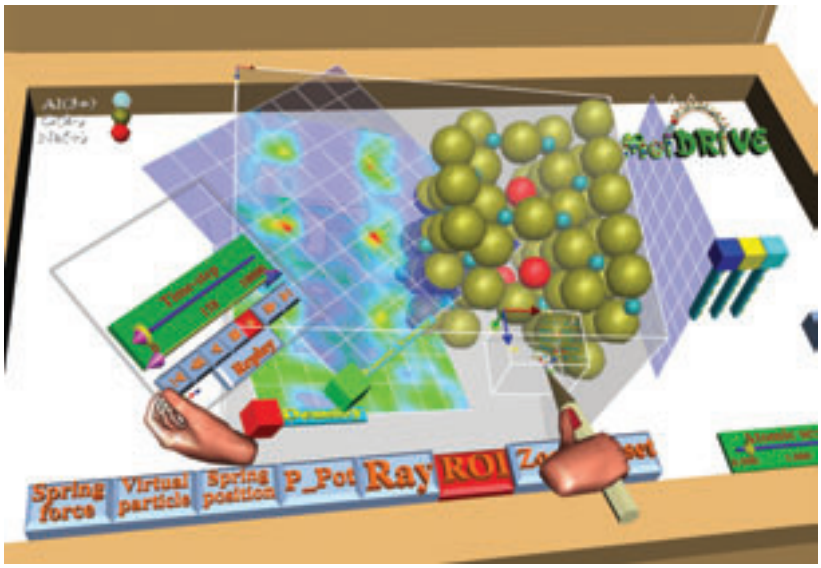
Color Figure 4: MolDRIVE (Gromacs): particle steering of protein fragment (see also Fig. 6.28 on p. 189)



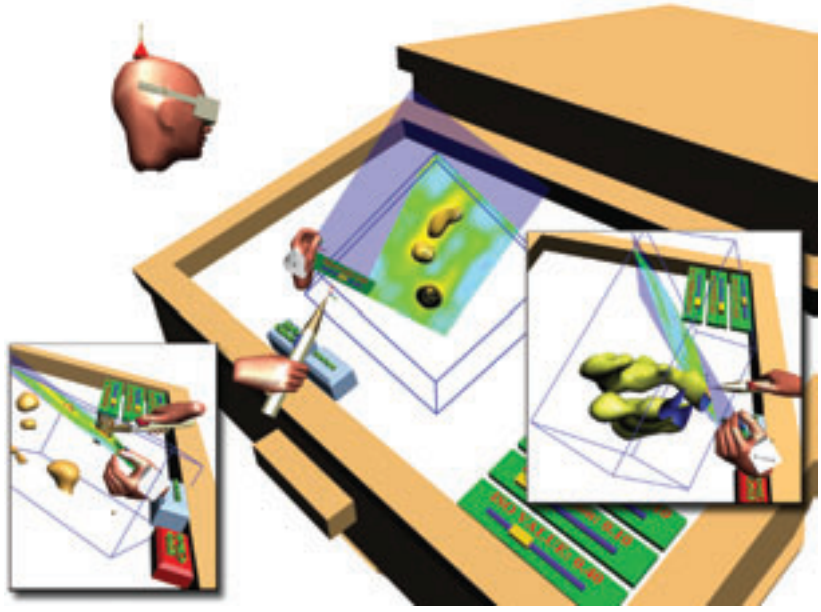
Color Figure 5: MolDRIVE (DEMMPSI): particle steering of the β-Alumina electrolyte with the spring manipulator on the RWB (see also Fig. 4.47 on p. 115)



Color Figure 6: MolDRIVE (Gromacs): particle steering of protein fragment with the spring manipulator on the RWB (see also Fig. 5.15 on p. 135)

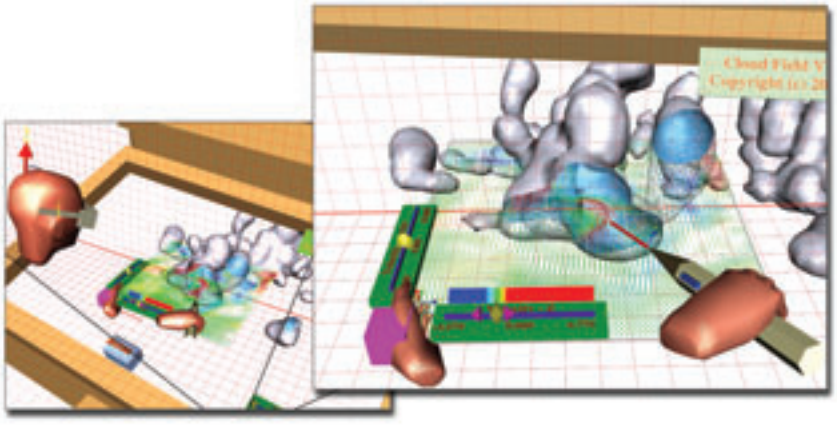


Color Figure 7: Visualization Client of MolDRIVE: time-control widget on the Plexipad; region-of-interest, mini system and several data slicers demonstrated in action (see also Fig. 6.21 on p. 184)

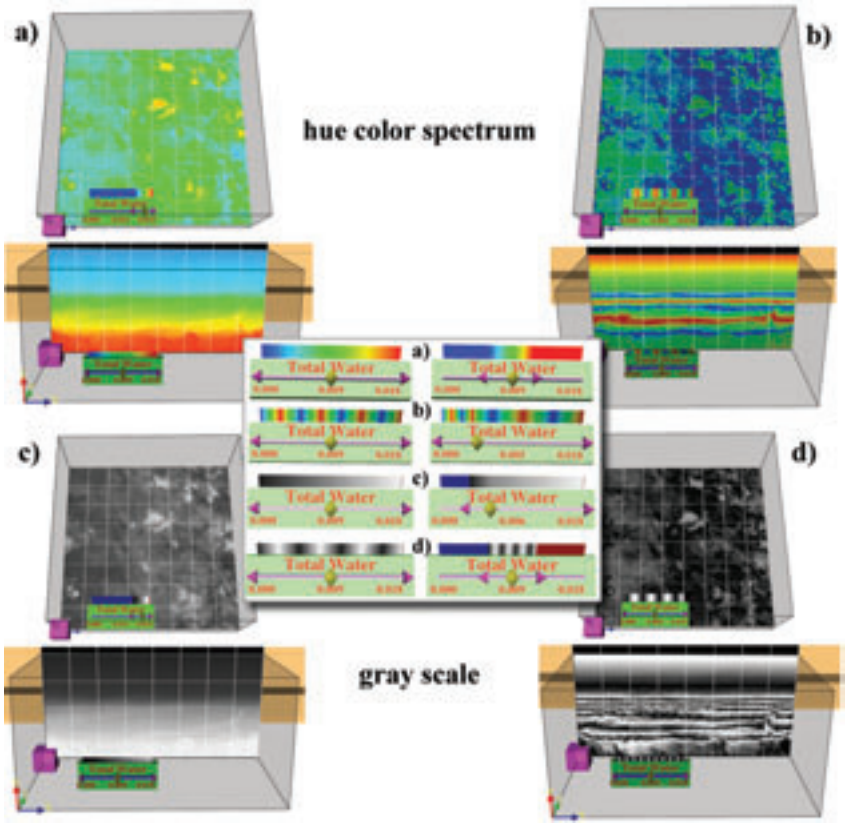


Color Figure 8: VRX Click-iso-surface tool: the data value on the selected point (on the Plexipad) is used as the input for generating an iso-surface (see also Fig. 5.13 on p. 132).

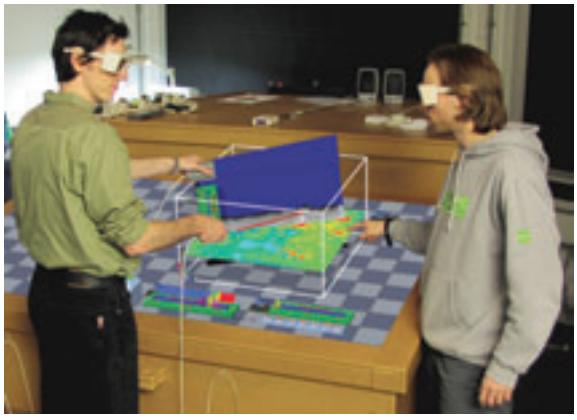
Color Figure 9: VRX Scalar data slicer and Geometry clipper: cumulus clouds (see also Fig. 6.50 on p. 212 and Fig. 6.54 on p. 215)



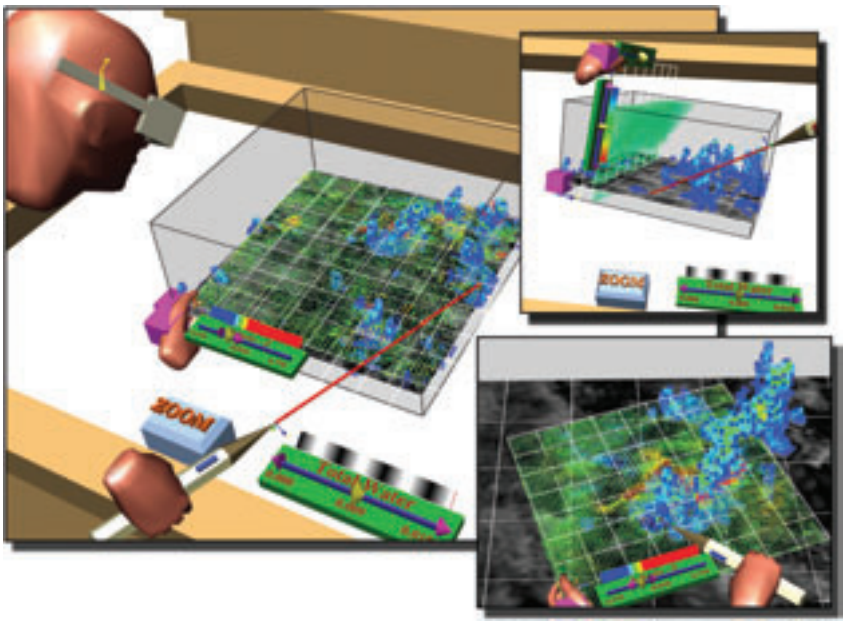
Color Figure 10: VRX Vector data slicer and Geometry clipper: cumulus clouds (see also Fig. 6.49 on p. 211)



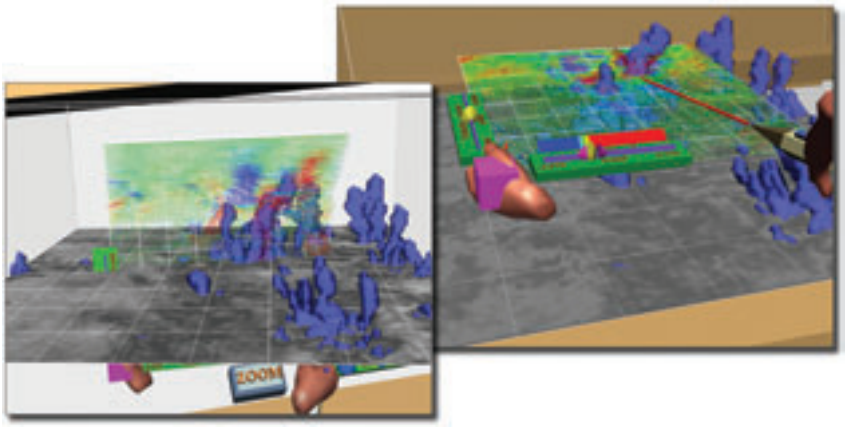
Color Figure 11: Basic color mapping scenarios in VRX: rainbow spectrum (a), periodic rainbow spectrum (b), gray scale (c), periodic gray scale (d); (see also Fig. 6.51 on p. 213)



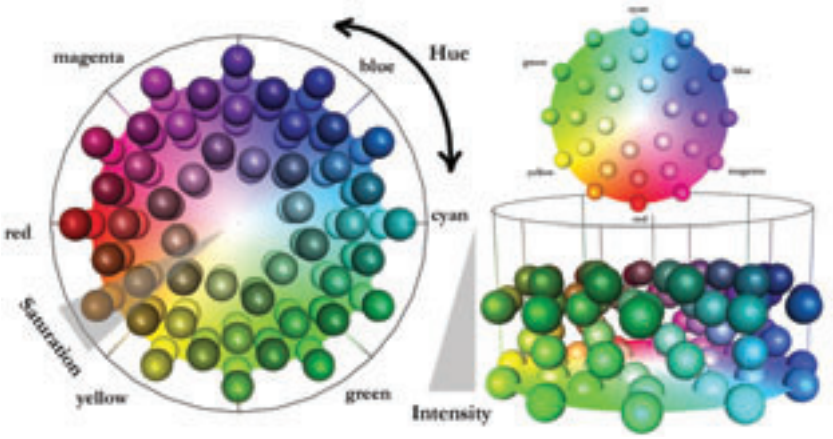
Color Figure 12: VRX Scalar data slicer tool: two-handed exploration of data (see also Fig. 5.1 on p. 119)



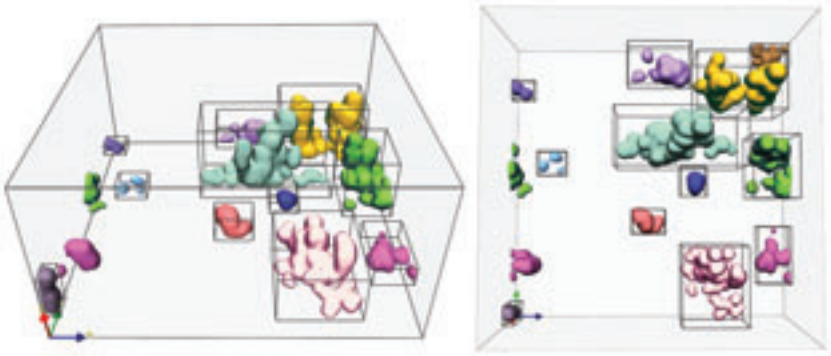
Color Figure 13: The direct vector data slicer is showing air velocity vectors colored with vertical velocity. Point-based volume rendering shows the q_l data (liquid water). The horizontal data slicer (gray-scale color mapper) shows the q_t data (total water). On this slicer constraints are applied so that it can move only in the Z direction, while the vector slicer can freely move with the Plexipad (see also Fig. 6.45 on p. 209).



Color Figure 14: Plexipad contains a direct vector data slicer, showing air velocity vectors colored with vertical velocity. The iso-surfaces show the cloud volumes (see also Fig. 6.46 on p. 209).



Color Figure 15: Optimized color scheme (bottom) with 96 different colors in the HSI color space (see also Fig. 6.43 on p. 207)



Color Figure 16: Coloring of cloud clusters: the clustering method also supports periodic boundaries (see also Fig. 6.41 on p. 206).

Color Figure 17: User-assisted tracking on the Virtual Workbench (see also Fig. 6.42 on p. 207)

Summary

Scientific visualization is a well-established method for analysis of data, originating from scientific computations, simulations or measurements. Due to the rapid progress in computer graphics and Virtual Reality (VR), we can see today a number of serious VR applications. Among applications in science also belongs visualization of data in virtual environments (VEs).

The potential of VR for three-dimensional visualization seems obvious. However, practically it is still very difficult to interact with virtual environments. Development of VR applications and design of VEs are also issues that need more attention. Visualization and exploration of data in VEs requires development of new visualization concepts different from those known from desktop workstations. These problems became motivation for this research project and this thesis.

The practical VR implementation has been performed on the Responsive Workbench (RWB). During the project we developed the RWB Library and RWB Simulator, our application software development environment. On top of this we designed and implemented VRX, an interactive visualization toolkit for the Workbench.

Chapter 2 contains a survey of related work in the field of visualization in VR, describing general principles of scientific visualization, the state of the art in Virtual Reality, concentrating on the research issues of scientific data visualization by means of VR. This chapter also gives a motivation of our work and outlines the research agenda of this thesis.

Chapter 3 introduces the Virtual (or Responsive) Workbench, also briefly describing its technical aspects. Further, the design aspects of applications and VEs for the Workbench are discussed. Interesting aspects are the laboratory table metaphor and the window-on-world metaphor. The technical constraints and Workbench specifics are reflected in the RWB Library. The concepts and usage of this library are described in detail. Implications of the development of RWB applications with the RWB Simulator are also discussed in the chapter.

Chapter 4 deals with interaction techniques in virtual environments. After related work in this field, interaction techniques specifically suitable for the Workbench are presented. This includes selection and manipulation of virtual objects, and navigation and exploration of VEs. The VR aspects of object collisions and constraints during object manipulation are also discussed.

The problem of providing force feedback on the Virtual Workbench has been addressed by a force feedback method for manipulation of virtual objects that can be used as an alternative to haptics. The spring-based manipulation tools (spring, spring-fork and spring-probe) are introduced. These tools are based on the spring metaphor, providing visual force feedback. The spring tools have been further adapted into the spring manipulator, which has been used for particle steering (manipulation with atomic particles) in a real-time Molecular Dynamics simulation, visualized on the Workbench using the MolDRIVE system. Three methods for particle steering are described and demonstrated with example applications.

Chapter 5 presents our approach to exploration and data visualization in VR. The description begins with our interaction and exploration tools. One- and two-handed interaction scenarios are discussed. The tools, derived from the scenarios, were based on several real-world metaphors, which made these tools very intuitive and easy to use. The navigation and probing tools developed were incorporated in the VRX toolkit, of which the concepts and techniques are also described in detail.

VRX is our modular object-oriented toolkit for exploratory data visualization, and it enables rapid development of visualization applications for the Workbench. A multiprocessing scheme and adaptive resolution of the probing tools is used for enhancement of performance.

Chapter 6 presents several case studies, dealing with visualization of scientific data on the Workbench. Three main case studies are described: interactive visualization of flooding scenarios, Molecular Dynamics visualization and computational steering, and visualization of cumulus clouds.

The cumulus clouds case study served a test-bed application during the development of VRX. For the Molecular Dynamics case study we developed MolDRIVE, a system for visualization and steering of MD simulations. This system has been used for studies of the β-Alumina electrolyte and several proteins.

This thesis shows which aspects are necessary for the development of virtual environments for visualization of scientific data. Interaction with the VEs is studied extensively. It is clearly demonstrated by object manipulation and by particle steering that visual force feedback, based on the spring metaphor, is a good alternative approach to the physical force feedback (haptics).

Samenvatting

Wetenschappelijke visualisatie is een algemeen geaccepteerde methode voor analyse van data afkomstig uit wetenschappelijke berekeningen, simulaties of metingen. Door de snelle vooruitgang in computer graphics en Virtual Reality (VR) zijn een aantal serieuze VR applicaties onstaan. Tot de applicaties in de wetenschap behoort ook de visualisatie van data in virtuele omgevingen (VEs - Virtual Environments).

Het potentieel van VR voor de 3D visualisatie lijkt overduidelijk. Echter, in de praktijk is het nog steeds moeilijk om interactief met de virtuele omgevingen te werken. De ontwikkeling van VR applicaties en het ontwerp van VEs zijn problemen die ook de aandacht trekken. Visualisatie en exploratie van data in VEs vereist de ontwikkeling van nieuwe visualisatie concepten die afwijken van de bekende concepten van desktop workstations. Deze problemen vormen de motivatie van dit onderzoeksproject en dit proefschrift.

De implementatie van experimentele VR omgeving is uitgevoerd op de Responsieve Workbench (RWB). Tijdens het project hebben wij de RWB Library en de RWB Simulator, onze software ontwikkel omgeving, ontworpen. Hierop hebben wij VRX ontworpen en geïmplementeerd, een interactieve visualisatie toolkit voor de Workbench.

Hoofdstuk 2 geeft een overzicht van verwant werk op het gebied van visualisatie in VR. Het beschrijft de algemene principes van wetenschappelijke visualisatie, de stand van de techniek (state-of-the-art) in Virtual Reality, waarbij wij ons concentreren op de onderzoeksvragen van de wetenschappelijke datavisualisatie met behulp van VR. Dit hoofdstuk beschrijft ook de motivatie van ons werk en schetst de onderzoeksagenda van dit proefschrift.

Hoofdstuk 3 introduceert de Virtual (of Responsive) Workbench en beschrijft ook kort de technische aspecten. Verder worden aspecten van het ontwerpen van applicaties en VEs voor de Workbench gepresenteerd. Interessante aspecten zijn de laboratorium-tafel metafoor en de window-on-world metafoor. De technische beperkingen en specifieke Workbench aspecten zijn terug te vinden in de RWB Library. De concepten in deze library en het gebruik ervan worden gedetailleerd beschreven. Verder worden in dit hoofdstuk implicaties van de ontwikkeling van RWB applicaties met de RWB Simulator behandeld.

Hoofdstuk 4 richt zich op interactietechnieken in virtuele omgevingen. Na het verwante werk op dit gebied, worden geschikte interactie technieken voor de RWB gepresenteerd. Daaronder verstaan we de selectie en manipulatie van virtuele objecten, en de navigatie in, de en exploratie van VEs. Tevens worden de aspecten in VR van botsende objecten en constraints op objecten tijdens de manipulatie behandeld.

Het probleem van het verschaffen van force feedback op de Virtuele Workbench wordt opgelost door een force feedback methode voor manipulatie van virtuele objecten die niet is gebaseerd op haptische of kracht terugkoppeling. Hiertoe worden de spring-based manipulatie tools (spring, spring-fork, springprobe) geïntroduceerd. Deze tools zijn gebaseerd op de veer (spring) metafoor, die zorg draagt voor een visuele force feedback. De spring tools zijn aangepast voor de spring manipulator, die wordt gebruikt voor de particle steering (manipulatie met atomaire deeltjes) in real-time Moleculaire Dynamica simulaties. Er worden drie methoden van particle steering beschreven en gedemonstreerd met behulp van voorbeeld applicaties.

Hoofdstuk 5 presenteert onze aanpak van de exploratie en de data visualisatie in VR. De beschrijving begint met onze interactie en exploratie hulpmiddelen (tools). Eén- en twee-handige interactie scenario's worden voorgesteld.

Deze tools, afgeleid van de scenario's, waren gebaseerd op metaforen uit de echte wereld, die ertoe leiden dat deze tools heel intuïtief en gemakkelijk te gebruiken zijn. De ontwikkelde hulpmiddelen voor navigatie en probing waren bevat in de VRX toolkit, waarvan het concept en de technieken ook uitvoerig beschreven worden.

VRX is onze modulaire object-georiënteerde toolkit voor exploratieve datavisualisatie; deze maakt een snelle ontwikkeling van visualisatie applicaties voor de Workbench mogelijk. Multiprocessing opzet en adaptieve resolutie van de probing tools worden gebruikt voor verbetering van de performance.

In hoofdstuk 6 worden enige case studies besproken, die handelen over visualisatie van wetenschappelijke data op de Workbench. Er worden drie hoofd case studies beschreven: de interactieve visualisatie van scenario's voor overstromingen, de visualisatie van moleculaire dynamica simulaties en computational steering, en tenslotte de visualisatie van simulaties van stapelwolken.

De case study van de stapelwolken diende als een proefapplicatie tijdens de ontwikkeling van VRX. Voor de moleculaire dynamica case study hebben wij MolDRIVE ontwikkeld, een systeem voor visualisatie en steering van MD simulaties. Dit systeem werd gebruikt in het onderzoek van het β-Alumina electrolyt en het eiwit onderzoek.

Dit proefschrift laat zien welke aspecten belangrijk zijn voor de ontwikkeling van virtuele omgevingen voor de visualisatie van wetenschappelijke data. De interactie met VEs is uitgebreid bestudeerd. Met behulp van object manipulatie en particle steering wordt gedemonstreerd dat de visuele force feedback, gebaseerd op de spring metafoor, een goede alternatieve aanpak is voor de fysieke kracht terugkoppeling (haptic force feedback).

Curriculum Vitæ

Michal Koutek was born on 19^{th} of November 1973 in Prague (the Czech Republic). In 1992, he received his Gymnasium diploma from Gymnázium U Libeňského zámečku (GULZ) in Prague. In 1998, he received his M.Sc. degree in electrical engineering and computer science from the Faculty of Electrical Engineering of the Czech Technical University in Prague. The title of his M.Sc. thesis was: "Optimizing motion of two legged figures". Using OpenGL graphics programming he developed an interactive system for inverse kinematics and inverse dynamics of animated two-legged figures.

In 1998, he started his PhD research project at the Computer Graphics and CAD/CAM group, the Faculty of Information Technology and Systems of Delft University of Technology. The research project involved interaction and data visualization in virtual environments. He also developed a basic software environment for applications on the Responsive Workbench. During his contract period, he has also actively contributed to the teaching activities of the group.

In 2002, he joined the Applied Physics Department of the Vrije Universiteit in Amsterdam, to conduct research on interaction with autonomous robots from computer generated environments.