Techniques and Architectures for 3D Interaction

About the cover

The cover depicts the embedding of 3D structures in both space and mind. At a quick glance or when viewed at an angle or curved page, the structures appear carefully organized. Closer inspection reveals that the shapes are all tilted at different angles. The background image was ray traced in the Blender 3D content creation suite, using subsurface scattering for the curved surface and ambient occlusion to emphasize structure. The final cover was typeset in Inkscape, with the BitStream Vera Serif font.

Techniques and Architectures for 3D Interaction

Proefschrift

ter verkrijging van de graad 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 woensdag 2 september 2009 om 10:00 uur

door

Gerwin DE HAAN

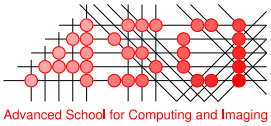
informatica ingenieur
geboren te Rotterdam.

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

Copromotor:
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, copromotor
Prof. dr. I.E.J.R. Heynderickx	Technische Universiteit Delft
Prof. dr. A. van Deursen	Technische Universiteit Delft
Prof. dr. ir. P.J.M. van Oosterom	Technische Universiteit Delft
Prof. dr. ir. R. van Liere	Technische Universiteit Eindhoven
Prof. dr. B. Fröhlich	Bauhaus-Universität Weimar



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 180.

Part of this research has been funded by the Dutch BSIK/BRICKS project (MSV2).

ISBN 978-90-8559-554-0

http://visualization.tudelft.nl/GerwinDeHaan

Preface

With the advent of new 3D simulation and measurements techniques, the demand rises for visualization of the generated spatial data. Much research in computer science focusses on advanced data processing, fusion and rendering. An intuitive 3D user interface should allow the user to have an effortless interactive exploration experience of these complex 3D datasets. A Virtual Reality (VR) system, traditionally used for intuitive exploration of hand-crafted virtual worlds, seems a logical candidate for these applications. We experienced however, that the 3D user interaction techniques in VR systems are often very basic and not directly suitable for every type of exploration task or dataset.

The original goal of this research was to investigate new interaction metaphors to support better 3D exploration tasks. In this thesis we first present improvements on a set of basic 3D user interaction techniques and interface elements. While working on the design and development of these techniques, we found that their implementation is a tedious and time-consuming task. Although general design concepts can be thought of quickly, the actual implementation and integration in several applications was often error-prone, and it took us much time and effort to adapt and combine techniques. During this project, our research efforts shifted to the cyclic development process of the VR application and their 3D interaction techniques. As a result of this, we present a software architecture and a modeling mechanism for prototyping new interaction metaphors.

From my work in this area, I have come to appreciate that the design of 3D user interfaces is an interdisciplinary area. Experts from computer graphics, visual perception, usability, hardware design and software engineering and domain experts of the datasets should be brought together in designing interfaces and interaction metaphors. This makes this an diverse and interesting field to work in, but at the same time difficult to position research and engineering activities. This diversity of interests is also reflected in the fact that some of my work could not be completely integrated within this thesis. This includes a mouse-based interaction technique for navigating surveillance videos in a VE [de Haan 09b], the PDRIVE VR system we designed [de Haan 07a], exploring the *Wii balance board* as a VR input device [de Haan 08b], and recent work on exploring large 3D pointclouds from aerial laser scanning. These items are discussed briefly in section 7.3 of the final chapter. Inter-

activity is approached from various points of view, and have different vocabularies and methods for discussing its features in many areas of expertise. Hopefully, our developer-oriented prototyping approaches can be extended to further ease the communication between various parties. In future, I hope to extend cross-disciplinary research to further support design and development of 3D interaction techniques.

The funding for this PhD project was provided by the Dutch BSIK/BRICKS research program (Basic Research in Informatics for Creating the Knowledge Society). As a part of this program, the research was conducted within the project *Modelling, Simulation and Visualization (MSV2): Interactive Virtual Environments*. The work for this dissertation was performed between 2004 and 2009 at the Computer Graphics and CAD/CAM group, part of the department of MediaMatics at the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), Delft University of Technology, The Netherlands.

In the acknowledgements section at the end of this thesis a list of people is included with whom I worked with during this period. Already here I would like to express my gratitude to everyone in our research group and all those who joined, guided or just greeted me during this project.

Gerwin de Haan

Delft, May 2009

Contents

Preface **v**

1 Introduction and Motivation **1**
1.1 Interactive 3D Visualization 1
1.2 Virtual Reality for Visualization 3
1.2.1 Data Visualization Characteristics 3
1.2.2 VR-Vis Application Examples 4
1.3 Acceptance of Virtual Reality and 3D Interaction 6
1.4 Software for VR-Vis applications 7
1.4.1 Interactive 3D Graphics Toolkits 7
1.4.2 Interactive 3D Visualization Applications 8
1.4.3 Interactive Scripting 10
1.4.4 Towards Interactive Virtual Environments 11
1.5 Techniques and Architectures for 3D Interaction 11
1.5.1 Designing Techniques for VR Characteristics 12
1.5.2 Iterative Development of Applications and Interaction 13
1.6 Thesis Content 14

I 3D Interaction Techniques **17**

2 IntenSelect: Assisting 3D Object Selection **19**
Overview 19
2.1 Introduction 20
2.2 Problem Analysis 22
2.2.1 Selection accuracy: Small and remote objects 22
2.2.2 Selection Ambiguity: Occlusion and Cluttering 22
2.2.3 Selection Complexity: Moving Objects 22
2.3 Related Work 23
2.4 Selection Algorithm 25
2.4.1 Overview 25
2.4.2 Selection Volume Test 25
2.4.3 Score Contribution 26

2.4.4 Score Accumulation . . . 27
2.4.5 User Feedback . . . 28
2.4.6 Flexibility and Extensibility . . . 30
2.5 Implementation . . . 31
2.6 User study . . . 32
2.6.1 Test setup . . . 32
2.6.2 Test Results . . . 33
2.7 Discussion and Future Work . . . 35

3 Hybrid Interfaces in Virtual Environments 37
Overview . . . 37
3.1 Introduction and Motivation . . . 38
3.2 Related work . . . 39
3.3 Windows and Widgets . . . 40
3.3.1 Windows . . . 41
3.3.2 Widgets . . . 42
3.3.3 Dialogs . . . 43
3.3.4 The Graph Window . . . 43
3.4 Supporting Interaction in VR . . . 44
3.4.1 Transition between direct and remote interaction . . . 44
3.4.2 Snapping behavior . . . 45
3.4.3 Object Manipulation . . . 47
3.4.4 Scoring Response control . . . 47
3.4.5 Scoring redistribution . . . 49
3.5 Results: Integrating Interaction and Interface . . . 50
3.5.1 VR System Characteristics . . . 50
3.5.2 Snapping and Constraints . . . 50
3.5.3 Selection and Readability . . . 51
3.5.4 Integration with Cloud Explorer . . . 52
3.6 Conclusions and Future Work . . . 52

4 Consistent Multi-User Viewing and Interaction 55
Overview . . . 55
4.1 Introduction . . . 56
4.2 Related Work . . . 57
4.3 Analysis and Approach . . . 59
4.3.1 Problem Description . . . 59
4.3.2 Alternative Camera Models . . . 60
4.3.3 Viewpoint Compensation . . . 61
4.4 Method . . . 62
4.4.1 Consistent Viewing . . . 63
4.4.2 Consistent Interaction with Scene Objects . . . 65
4.5 Evaluation . . . 67

4.5.1 Experiments . . . 67
4.5.2 Conditions . . . 68
4.5.3 Results . . . 69
4.6 Conclusions and Discussion . . . 72

II Architectures for 3D Interaction 73

5 Interactive VR Software Prototyping 75
Overview . . . 75
5.1 Introduction . . . 76
5.2 Related Work . . . 77
5.3 Prototype Description . . . 79
5.3.1 Software Layers . . . 79
5.3.2 Wrapping of existing software components . . . 79
5.3.3 Control Beyond Wrapping . . . 81
5.4 Prototype Results . . . 81
5.4.1 Run-time Prototyping . . . 81
5.4.2 Internal Extensions . . . 82
5.4.3 External Software Integration . . . 82
5.4.4 Iterative Development . . . 84
5.5 Conclusions and Future Work . . . 85

6 StateStream Model and Architecture 87
Overview . . . 87
6.1 Introduction . . . 88
6.2 Related Work . . . 90
6.2.1 Model-based Design . . . 90
6.2.2 Model Integration . . . 92
6.2.3 Development Environment . . . 93
6.3 Model Description . . . 93
6.3.1 Description Language . . . 94
6.3.2 Actor Domain . . . 95
6.3.3 Discrete Domain . . . 95
6.3.4 Continuous Domain . . . 95
6.3.5 Integration . . . 97
6.4 StateStream Prototype . . . 98
6.4.1 Base Architecture . . . 98
6.4.2 StateStream integration . . . 99
6.4.3 Front-End Interface . . . 99
6.5 Results . . . 100
6.5.1 Selection and Manipulation . . . 100
6.5.2 Multiple Object Selection . . . 102

6.5.3 Snap Measurements . . . 103
6.5.4 Two-Handed Scaling . . . 103
6.5.5 Development Use . . . 104
6.6 Discussion . . . 104
6.7 Conclusions and Future Work . . . 105

7 Conclusions and Discussions **107**
7.1 Research Questions and Contributions . . . 107
7.2 Current State of the Work . . . 109
7.2.1 Interaction techniques . . . 109
7.2.2 Architectures for Interaction . . . 111
7.3 New 3D Devices and Applications . . . 112
7.4 Software Development and Maintenance . . . 115
7.5 Future Work . . . 115
7.6 Vision on Interactive 3D Exploration . . . 117

Bibliography **119**

Summary **127**

Samenvatting **129**

Curriculum Vitae **131**

Acknowledgements **133**

1 Introduction and Motivation

This thesis concerns three-dimensional (3D) interaction techniques with computer systems. Our main interest in 3D interaction is its use and development in the context of interactive 3D data visualization. In this chapter, we will first briefly discuss concepts of visualization and Virtual Reality, and then discuss the role of 3D interaction in this context. We discuss the motivation for our work, describe the background in the research area and present a research agenda. Finally, we present an outline of this dissertation and a list of the contributions.

1.1 Interactive 3D Visualization

The world's interest in complex, three-dimensional shapes and structures of many phenomena and objects is fed by recent technological advances. Scientists and engineers increasingly use advanced tools for computer simulation and data acquisition to collect massive amounts of spatial data for analysis. Examples of these are simulations of the atmosphere, weather and clouds and the measurements of coast, dunes and dikes with radar and laser range scanning techniques [Shan 08]. Computer generated, interactive 3D visualization can be used to explore these datasets to look for new information and to gain insight.

Highly detailed 3D visualization can be performed on today's computers. In the field of computer graphics and visualization, much work addresses issues of performance of data handling, data processing and rendering [Johnson 04]. This is needed to keep up with the ever growing resolution and complexity of simulations

Figure 1.1: Inspecting a toy car as an analogue to 3D data exploration. Imagine exploring the toy car from static pictures only (left), when on display in the toystore (middle) or when held in your hands (right).

and acquisition techniques. Many ongoing efforts focus on designing display and visualization algorithms with high detail at *interactive frame rates*. This is to maintain sufficient reactivity to the user's commands, also with these larger datasets. For an effortless exploration however, a system's fluent reactivity is useful only when combined with an "intuitive" user interface.

This intuitiveness in exploration of a 3D dataset can be illustrated by the analogue of inspecting a toy car in the real world, see Figure 1.1. First, if only a few photographs of the toy car are available from a store catalog or web site, it is hard to get a good impression of all the details, dimensions and spatial relations. If the car would be on display in the shop window, you could get a better sense by moving closer and trying to look around it. If the car would be right there on the shelf, you would pick up the car in one hand, rotate it and use your other hand to try and open its doors or turn the steering wheel. In the first two cases, restrictions in the display modality and interaction limit the information and freedom of exploration. In the last case, you use skilled hand-eye coordination to inspect the object, and even manipulate parts of it in a natural way.

When working on a desktop computer, the visualization software presents its 3D images of a dataset on a regular 2D display monitor. Users control the exploration with a mouse and keyboard and a set of standard interface elements. This is often sufficient for basic exploration of many datasets. For those datasets which contain complex 3D shapes and structures, and whose exploration requires spatial interaction tasks, it may be too restrictive; For these tasks, the awkward interaction controls lack intuitiveness, thereby continuously interrupting the exploration process.

In these cases, the lack of 3D display and 3D input make interactive 3D exploration and 3D manipulation cumbersome. Ideally, the full "bandwidth" of human perceptual and motor skills from the real world should be used for inspecting virtual 3D datasets.

1.2 Virtual Reality for Visualization

Virtual Reality (VR) systems are useful for exploring virtual worlds because of their intuitiveness in spatial perception and interaction. These systems are available in many forms, but in general they combine a stereoscopic display with spatial tracking of the head and hand-held devices, for an overview see [Burdea 03]. They are most famous for their use in psychology and entertainment applications, where a user wears a head-mounted display to be *immersed* in a computer generated world. Virtual Reality systems can also be used for data visualization. The use of stereoscopic displays combined with tracked 3D input devices and interaction tools allows a user to study a dataset in the virtual world with ease. For an overview of several Virtual Reality Visualization (VR-Vis) applications, see [van Dam 00].

1.2.1 Data Visualization Characteristics

Data visualization applications in VR have different characteristics from other VR applications. This difference mainly originates from the type of data and the interaction during visual exploration. In many Virtual Environments, such as in the case in viewing a car model or an architectural walk-through, the scene and the objects are created from 3D models. The shape, color and texture of an object is explicitly modelled in geometric primitives (e.g. points, lines and surfaces) to form a visual 3D model. A strong focus is on direct perceptual realism, in terms of the realistic visual rendering of the 3D models.

In data visualization applications however, the visual elements are often automatically generated from the raw measurement data or simulations itself. The *Visualization Handbook* [Johnson 04] gives a good overview of the current state of the art in data visualization. This occurs in a process called the *visualization pipeline*. In this pipeline of data operations, raw data is *filtered* first, then *mapped* to geometric shapes and their attributes (e.g. color), which are finally *rendered* visually. This process may also be executed dynamically, such as in the case of direct coupling of live simulation results to a visualization, e.g. see the MolDRIVE application in section 1.2.2.

The resulting abstract, visual elements such as *iso-surfaces* or *particles* may not have a physical form or are not visible in the real world. The focus here is not directly on *visual realism*, but instead on the *physical correctness* of the representation of the data. Visualization aims to maximize the amount of information and its comprehensibility that can be condensed into (abstract) visual representations. The strength of information transfer of a visualization application lies in the optimal use of the visualization technique, and the interactive controls it provides to change these visualization tools and their parameters.

The interactivity in data visualization applications takes place in the operations of the visualization pipeline itself, and in the interactive exploration of the generated visual output. In desktop-based visualization applications, interaction is predominantly on the details of the data visualization pipeline itself. This includes the

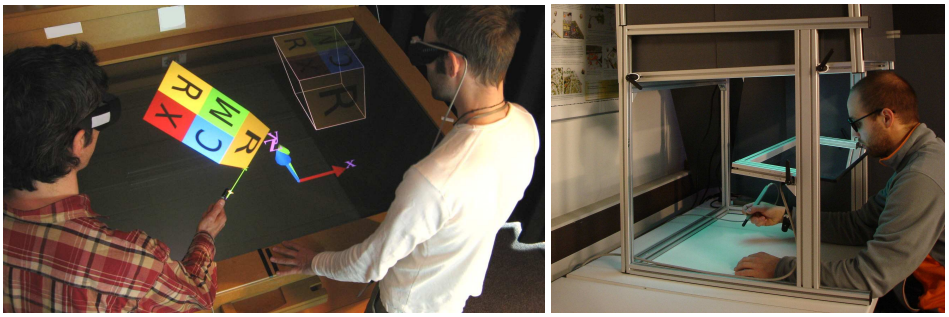


Figure 1.2: The active space VR systems in our VR Laboratory: The Responsive Workbench (left) and the PDRIVE [de Haan 07a] (right).

iterative process of selecting methods for data filtering and visualization algorithms and setting their parameters. In contrast, VR-Vis environments are best suited to the exploration of the spatial structures and the underlying phenomena [Bryson 96]. To accommodate this exploration (3D) interaction mechanisms are employed. Also in terms of interaction mechanisms, VR-Vis has different characteristics from other VR applications. In contrast with many other VR navigation tasks, navigation in visualization is mainly not directly related to first person or ego-centric, spatial wayfinding and travel through the virtual world [Bowman 04]. Instead, the visualization data domain or "box" object comprises the virtual world. Also, it can contain the visualization elements, such as a particle emitter, or a region of interest. These objects are inspected externally or exo-centric, which turns spatial navigation into the selecting and manipulation of these objects in space. These objects should be created and manipulated in 3D space. Abstract parameters such as the setting of a parameter value need to be made available as well. For this, the use of abstract controls in the 3D world such as widgets are necessary.

1.2.2 VR-Vis Application Examples

In earlier work in the TU Delft Data Visualization group we have gained experience in building and using interactive 3D visualization applications in VR [Koutek 03]. Here, we mainly use semi-immersive, projection-based VR systems such as the Responsive Workbench (RWB) or the PDRIVE, see Figure 1.2. The RWB [Krüger 95] is a tabletop VR system and the PDRIVE [de Haan 07a] is a desktop-based VR system, see also section 7.3. In both systems, a Polhemus electromagnetic tracking system is used to track interaction devices. These systems are also called *active space* or *reach-in* interaction systems, or even a virtual laboratory table in the context of data visualization. This is because the user is not fully immersed in the virtual world, but he does use his hand-eye coordination while reaching in the Virtual Environment (VE)

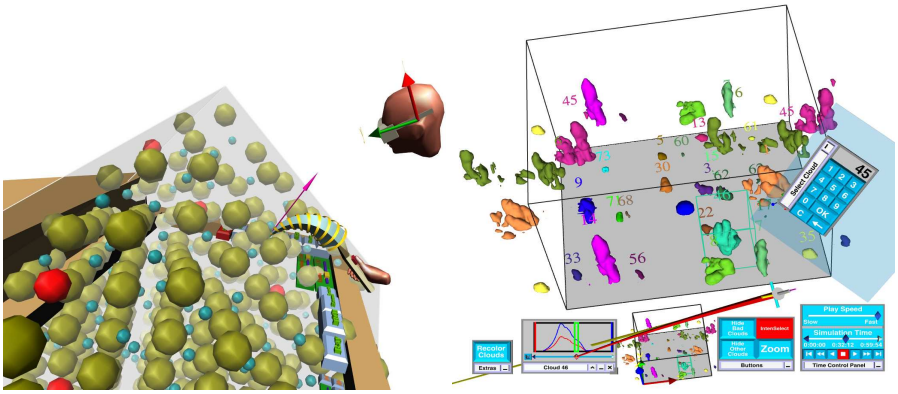


Figure 1.3: Impressions of two interactive visualization applications in VR: MolDRIVE [Koutek 02] (left) and Cloud Explorer [Griffith 05] (right).

where the data is displayed. A correct calibration ensures that the 3D position of the tracked input devices matches the 3D visual representations seen in the VE, co-location, is achieved. With respect to the toy car example, this type of VR system can be best compared to the "hold in hand" scenario.

Two examples of our visualization applications in Virtual Environments are MolDRIVE and Cloud Explorer. These applications focus on the visualization and control of (real-time) simulations of physical processes.

The MolDRIVE system allows researchers to explore Molecular Dynamics (MD) simulations at run-time of various 3D molecular structures such as electrolytes, polymers, and proteins [Koutek 02, Koutek 03]. We adapted existing MD simulation packages to run on separate machines and to continuously transmit simulation updates over the network to a running visualization application. The visualization runs on the VR system, and displays the individual atoms and their properties –as they are simulated– in the Virtual Environment. Using 3D interaction tools, the user can select and steer individual atoms or display the field properties of the simulation, see Figure 1.3 (left). An overview of this set of exploration tools for visualization in VR is given in [de Haan 02].

Cloud Explorer is an application under development for analysis of atmospheric simulation data, for example life-cycle studies of cumulus clouds [Griffith 05]. First, atmospheric simulations generate large data sets, from which various information modalities and features are extracted in an off-line preprocessing phase. The resulting data is visualized in the VE, which allows one to browse through time and inspect 3D features such as the cumulus clouds, see Figure 1.3 (right).

1.3 Acceptance of Virtual Reality and 3D Interaction

The creation of new interaction techniques to *"interactively manipulate and explore data"* has been stated as one of the grand challenges in the field of data visualization [Munzner 06]. From our experience in building and working with VR systems, we believe its characteristic 3D user interfaces (3DUIs) will become essential elements for the more intuitive, interactive exploration of complex 3D datasets. In practice however, there are relatively few new visualization applications –and extensions of existing ones– that employ VR systems. We consider the following two barriers to largely contribute to this lack of acceptance.

First, large investments in VR-specific hardware are needed. The required display and input hardware for creating a VR system is not standard. Because of this, cost and availability is an issue in the adoption. Under pressure of innovations and trends in consumer entertainment markets, we see more displays and input devices becoming cheaper and of higher quality. We have performed some work on this issue in the design and construction of the PDRIVE system, as shown in Figure 1.2. It is built from relatively cheap commercial-off-the-shelf components and standard construction material and can be assembled in a day. As this is beyond the scope of this thesis, we refer to the article for more details [de Haan 07a].

A more important issue is that labor intensive investments in VR-specific software are needed. Existing 3D visualization and rendering algorithms can be adapted relatively quickly to produce correct, stereoscopic images. However, most existing user interaction in visualization systems heavily relies on Graphical User Interfaces (GUIs) designed for 2D desktop computing. The use of normal Windows, Menu and Pointer (WIMP) interfaces in a Virtual Environment does not result in a usable environment. A transition is required from a desktop, mouse- and keyboard-based interface to a 3DUI controlled by spatial input devices.

Because of the lack of the standard interfaces, the emphasis in exploring visual data is on 3D interaction. When building these VR applications however, we noticed it was not easy to incorporate new 3D interaction techniques within existing visualization packages or graphics toolkits. During development, much of the (static) data visualization pipeline can be set up. During 3D exploration, one would want to switch visualization styles and use direct 3D input to adjust parameters. However, this is not often directly feasible within the 3D world. The functionality needs to be "naturally" mapped to 3D pointing devices and minimal mode switching. As these resulting interfaces need more attention on robustness and intuitiveness, their development and design requires much effort.

In the remainder of this chapter we discuss related work in the research area of 3D interaction. We observe interaction issues from the perspective of a user, discussing usability of basic 3D interaction in a VR system. Then we observe the issues from the perspective of a developer, discussing how a software architecture can support the development cycle for VR applications and applying more advanced interaction techniques. From these observations we present a research agenda and give an

overview of the contributions of this thesis.

1.4 Software for VR-Vis applications

The creation of interactive visualization applications in a VR context involves combining a heterogeneous set of software components, such as rendering, data visualization and tracking [Bryson 96]. In the following subsections we will briefly describe the first two components, and how customization and interactivity within these components is defined.

1.4.1 Interactive 3D Graphics Toolkits

The most prominent component of the VR software architecture is the graphical rendering component. In interactive 3D graphics, low-level graphics APIs with high performance such as *OpenGL* [Shreiner 05] have been used. To obtain more manageable systems, often a higher level 3D graphics toolkit is used. Historically, two of these toolkits can be considered essential for the development of VR architectures: IRIS Inventor and IRIS Performer.

In the early 1990's, Silicon Graphics Inc. (SGI) introduced *IRIS Inventor* to provide an object-oriented 3D toolkit to enhance productivity of graphics programming [Strauss 93]. It shifts the graphics programming paradigm from creating explicit, immediate-mode 3D drawings to creating implicit or retained-mode 3D objects. These 3D objects are stored in a scene database, the hierarchical *scene graph*. The toolkit provides a library of 3D objects, 3D manipulator widgets and application components such as a material editor and a 3D viewer, see Figure 1.4. It also provides a 3D interchange file format, of which many features have been adopted by the *VRML* standard [Carey 97] and its more recent successor *X3D* [x3d 07]. Iris Inventor and its successor Open Inventor was closed source and commercially licensed to vendors such as TGS. After the source code for Open Inventor had been released in 2000, it has been adopted by several other applications and toolkits. For example, its concepts are recreated in the *Coin3D*[1] system.

Where IRIS Inventor originally focussed on ease-of-use and programming in 3D graphics, SGI targeted an alternative toolkit *IRIS Performer* [Rohlf 94] to focus on high-performance rendering on their hardware. This system is also based on the scene graph concept, but it does not include the many end-user components and manipulator widgets that made Inventor easy-to-use. Instead, it was designed to provide "hard" real-time graphics for visual simulation, such as flight simulators. To achieve this, its features include multiple graphics outputs, multiprocessing, database paging and level-of-detail representations. For this reason of

[1] http://www.coin3d.org

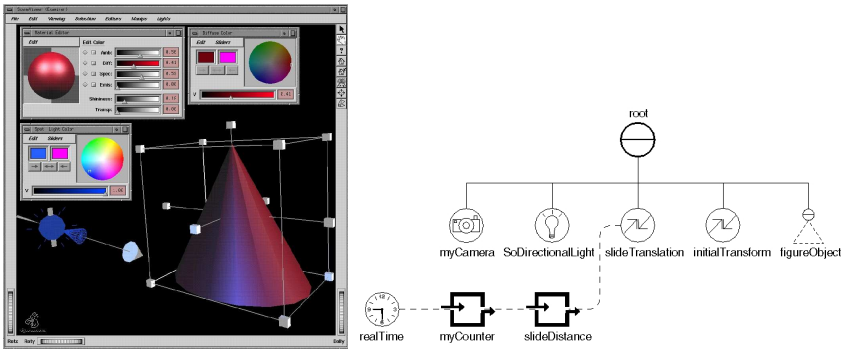


Figure 1.4: IRIS/Open Inventor illustrations. Left: Screenshots of window, consisting of a viewer, dialogs and 3D manipulation widgets. Right: A scene graph with data flow, where a timer is connected to a translation of a node (from [Wernecke 93]).

high-performance, many Virtual Reality systems were originally built on IRIS Performer and its successor OpenGL Performer, see [Koutek 03]. More recently, open source toolkits such as OpenSceneGraph[2] and OpenSG [Reiners 02] adopted the main scene graph concepts and now form the basis for many VR toolkits.

The essence of interactive graphics systems is the ability to customize interaction mechanisms of objects in the scene. Several software design mechanisms are provided to enable this customization. In the scene graph systems, nodes can be equipped with callback mechanisms to accommodate custom code for handling user input events and dynamic animation. In Open Inventor, also a mechanism for basic data flow is provided. This mechanism defines relations between nodes, and is conceptually orthogonal to the scene graph. For example, one can connect "fields" of a node to another field in another node, see Figure 1.4, right. A field connection can also be passed to an Engine node which "filters" data, e.g. an engine can add a vector to a 3D location to constrain one shape to another with an offset by this vector. With sensor and event nodes, user events can be propagated through the network. To ensure a correct data flow execution, it is required that the evaluation of nodes and connections is executed in the right sorting order. However, this correct creation of order and solving cycles of dependency can be problematic.

1.4.2 Interactive 3D Visualization Applications

Architectures for data visualization focus on the processing of data and the mapping of data to graphical primitives. Therefore, many of these are data-oriented and follow a data flow mechanism. The individual processing or filtering components are connected programmatically through an API or visually in a visual network ed-

[2] http://www.openscenegraph.org

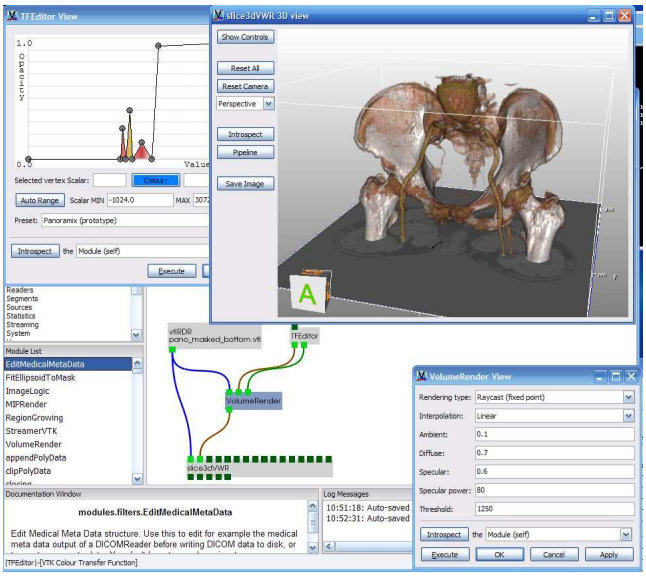


Figure 1.5: Screenshots of the DeVIDE visualization applications [Botha 04].

itor, see Figure 1.5. Examples are visualization application builders such as AVS [Upson 89] and OpenDX[3]. Programming toolkits such as the Visualization Toolkit (VTK) [Schroeder 06] provide an API to construct visualization software. Other visualization applications, for example DeVIDE [Botha 04], build on this toolkit functionality.

In data visualization systems, the interaction with the data and the visual representation can be done either via setting a property directly or via direct manipulation in the graphics canvas. Setting a value is done by using the programming language through an API, or through graphical control components in dialog windows, e.g. using a slider widget to update a value. In this case, the interaction changes values, which propagate through the data flow network. Direct manipulation is done with the mouse cursor directly in the graphics canvas, such as 3D view navigation, selecting points on a surface, translating and rotating elements. In a fashion similar to Open Inventor, often 3D manipulators or widgets are used to perform these controls. The direct feedback on manipulator changes are directly programmed in these manipulator and generally do not follow the data flow mechanism. Custom callbacks in the manipulators are used to reflect changes into the data flow network.

The two different approaches to interactivity (data flow updates, interaction updates) must be combined. An interesting approach is MeVislab[4], a platform for de-

[3]http://www.opendx.org
[4]http://www.mevislab.de/

veloping medical imaging applications for use in clinical environments. It combines Open Inventor mechanisms for scene control with data flow for the visualization data. However, the visualization modules themselves usually do not provide real-time feedback on changes in the input. The maximum update rate of a data flow pipeline is therefore often too slow (e.g. two frames per second) to provide direct feedback on interaction with the scene. Interactively controlled tools with direct visual updates, e.g. a 2D texture volume slicer, are often custom designed to circumvent the existing data flow mechanisms. This has the downside that feedback on interaction is not made explicit and is not visible in the data flow graph.

1.4.3 Interactive Scripting

Ousterhout already indicated the value of scripting capabilities [Ousterhout 98] as a higher level programming mechanism. In interactive 3D graphics systems, scripting was introduced to allow extensibility of interactivity without resorting to the restrictions of a scene-graph or a pure data flow mechanism. As a precursor to full-fledged scripting, the Inventor file format already allows simple scene-graph configuration constructions without resorting to C++ code. The VRML specification allows the use of *script nodes*, which can contain user code written in *VRMLscript*, *JavaScript* or *Java*. For IRIS Inventor, invenTcl [Fels 97] was presented and PiVy[5] used Python to provide bindings for Coin3D. Also the performance-oriented VR toolkits have included some form of flexible scripting. For example, Avango uses *Scheme* scripting and the Lightning Toolkit uses *Tcl* scripting. We maintain a list of VR toolkits with scripting capabilities on a web page[6]. In a similar fashion to the 3D Toolkits, many of the visualization applications and toolkits include scripting capabilities. For example VTK has the option to build a visualization data flow pipeline through the use of Tcl and Python-based scripts. The DeVIDE framework is of interest as the core of the environment is built in the scripting language and glues all external software components together [Botha 04].

The influence of the scripting extension depends first on the *hooks* in the code where one can script and what part of the architecture can be addressed. Also, their dynamics plays a role in the level of interactivity: Scripts can be precompiled, interpreted at run-time, or even dynamically used during run-time. However, the freedom of custom callbacks and scripting can interfere with -or even break- the general operation of the toolkit. At the same time, the flow of control within these scripts is not an explicit part of the model and might hide information on its operation.

[5] http://pivy.coin3d.org/
[6] http://visualisation.tudelft.nl/VRdev

1.4.4 Towards Interactive Virtual Environments

Most interactive graphics systems are designed with the desktop computer with mouse and keyboard in mind, not for the VR setting. This is reflected in the way interfaces and interaction tools are designed in these systems. Even user interfaces that allow 3D control via manipulators or explicit 3D widgets, are designed with 2D mouse input in mind. If applications based on these systems are transferred to a VR system, the potential benefits of VR are not automatically realized.

In general, applications do not provide much more than basic stereoscopic rendering. Only when 3D input devices and stereoscopic displays are supported, more VR functionality can be realized. Extension toolkits provide the added functionality to connect VR input and display devices to the graphics framework. An example of this is *VR Juggler* [Bierbaum 01], which glues together rendering libraries with tracking and audio subsystems. VR systems such as Avango [Tramberend 99] and Lightning [Blach 98] reintroduce the concept of fields and the data flow graph in OpenGL Performer. Other libraries focus on the interchange of input devices without reprogramming the application.

The basic interaction techniques such as basic six Degree-of-Freedom (6DOF) pointing and head tracking, selection and ray-casting intersections are usually provided by the VR framework. However, more sophisticated 3D interaction techniques on top of these, such as slicing planes and region selection are often missing. In general, only a small set of standard 3D interaction techniques is present that work over a wide range of applications and VR systems.

From previous work in the TU Delft Data Visualization group, the RWB library provided some of these basic interaction tools [Koutek 01a]. These were later extended to include spring-based manipulation tools [Koutek 01b]. Using the metaphor of bending a spring, these manipulation tools incorporate bending of the interaction tool as a visual indicator to give feedback on manipulations that are constraint. In our Virtual Reality eXplorer (VRX) toolkit, we included a set of data visualization specific interaction metaphors [de Haan 02]. It became clear that during the creation of these tools, the software architecture restricted flexible extensions of existing behavior. Much code was replicated for each technique, which involved lots of manual maintenance and introduced many unforeseen errors in interaction tools. As a result, we had to redesign the overall interaction architecture in several iterations.

1.5 Techniques and Architectures for 3D Interaction

The introduction of data flow and scripting on top of the scene graph abstraction has enabled more flexibility of application control. However, in many graphics and visualization applications the interaction and interfaces are still viewed as ”afterthoughts” and are programmed around –and seem orthogonal to– the visual ren-

dering. Therefore, a structured integration of flow of control and interaction often remains a tedious task.

To benefit from the features a Virtual Reality system offers, 3D data exploration tools have to be transferred and integrated into such system. In the case of VR systems, we feel interaction and 3DUIs should be primary components that deserve careful attention. In addition, support is needed in the process of design and development of these techniques and for a good integration with the application. This leads to the research agenda containing the following two themes on 3D Interaction: designing techniques with the possibilities and limitations of VR system characteristics in mind, and iterative development of applications and interaction. In the following two subsections we will elaborate on these two items and the research questions involved.

1.5.1 Designing Techniques for VR Characteristics

The first major observation discussed in this thesis is the fact that the "lack of precision" of the human perceptual and motor skills should be taken into account in the design of 3D viewing and interaction techniques for VR.

It is natural that a VR system does not produce a full, physical 3D replica of the data explored. Instead, with the help of stereoscopic displays and 3D tracking, we evoke the effect of observing –and interacting with– "believable" and "significant" 3D objects floating in space, or *"create the effect of interacting with things, not with pictures of things"*[Bryson 96].

This effect is sufficient, such that one can employ some real-world perceptual and action skills to work with objects in the virtual world intuitively. However, there are limitations to the level of intuitiveness that can be reached in this situation.

On the one hand, an essential element of a Virtual Environment is the lack of physical rules and restrictions. It allows one to freely configure the world, and also enables one to perform many interaction techniques that do not have direct real-world analogues. A simple example of this is selecting –and manipulating– an object at a distance. For users it takes a lot of effort to accurately select objects, especially in complex worlds. This is caused by the lack of accuracy in tracking and in human motor action (e.g. position tracking and pointing) and by the lack of physical feedback by the objects.

On the other hand, the method of display removes some basic physical aspects that we take for granted in the real world. This can introduce new problems, for instance if we try to extend the VR experience to multiple people. The single, head-tracked stereo view, causes other non head-tracked users to have a bad view on the 3D scene. We experienced that a user can tolerate and compensate for some distortion in viewing. However, the interaction with objects –intuitive for a single user– becomes difficult. This prevents a convincing collaborative experience in a 3D Virtual Environment.

In the current VR applications, the user is often considered to be a perfect observer (looking at a perfect 3D world) and a perfect actor (able to perfectly select and navigate in a 3D world). From the observations mentioned above, we know this is not the case in practice. We should strive to provide a more *stable* 3D interaction experience that is less sensitive to inaccuracies, distortions and misalignments in display and tracking and to inaccuracies in human hand motion and manipulations. This brings us to the first research question:

> *How can we develop 3D interaction techniques for use in a Virtual Environment that are "stable" with respect to inaccuracies in the display of objects, calibration of tracking devices, and human manual capabilities?*

In this thesis, we use these observations to redesign some aspects of 3D interaction. The goal is to reduce effort in interaction and to provide richer collaborative exploration in VR applications. We want to provide more context sensitive selection techniques and reduce of viewing distortion and its impact on collaboration through alternative camera models and view warping.

1.5.2 Iterative Development of Applications and Interaction

The second major observation in this thesis is, that in the development and design of VR applications and 3D interaction techniques, the underlying architecture should facilitate a high level of integration and allow adaptability.

As discussed in the previous sections, VR systems integrate 3D display, user interface and special device communication and involve a heterogeneous set of software libraries and toolkits. A wide variety of VR systems and applications with different characteristics exists. Often, these are custom solutions specifically geared towards one application and on one specific VR type of system. It is this relative uniqueness of the combination of VR hardware, custom application and high interactivity that makes it difficult to envision what the possibilities are and what will work in the most intuitive and insightful way. It is also for this reason that new applications, or even a new dataset requires careful attention. During the life-cycle of an application, changes are frequently suggested and/or introduced by a wide range of people involved, such as the end-user, the original developer, a user-interface specialist or an application domain-expert.

The design and development of VR applications should be flexible enough to accommodate these changes. At the same time, interactive 3D graphics applications require high performance algorithms, VR applications included. For this reason, they are generally developed using powerful graphics toolkits and low-level programming languages such as C++. The steep learning curve for these APIs and low-level languages form a barrier for wide involvement and change. A problem here is the strictness of the base application layer or framework, which is often too low-level to accommodate fast changes.

We observe in practice that we need many iterations of development, during which applications and interaction techniques slowly emerge. The software architecture needs to be equipped to better match the "emergent" style of development and design. Our goal is to achieve a more flexible development environment for creating interactive 3D applications. In this thesis, we approach this by providing more flexible layers of programming abstractions.

This emergent style of development and design especially affects 3D interaction techniques themselves. The 3D user interaction techniques fully integrate the input device actions in the 3D world. It is for this that 3DUIs require more integration with algorithms and better user feedback than regular desktop user interfaces. At the same time, these interaction techniques should be tuned and adapted to the specifics of the VR system and application in use. For the developers and designers, it is difficult to get an overview of the many possible situations.

The complexity of relations between individual 3D objects and the tools that operate on them can be overwhelming. Although prototyping flexibility is provided by programming layers, it does not provide structuring elements for taking interaction techniques to the next level. In terms of describing interactivity and (3D) interaction techniques for VR, we feel this has not introduced much structure.

Without a proper abstraction, the amount of relations and specific behavior that are needed quickly leads to increasing complexity. This brings us to the second research question:

> *How can we support generic prototyping for fast development of and testing of interactive 3D visualization applications and their interaction techniques in a Virtual Environment?*

In this thesis, we will discuss suitable modeling abstractions in the context of 3D interaction and how this facilitates prototyping of interaction tools.

1.6 Thesis Content

For clarity, the main content of this dissertation is divided in two parts. The chapters in each part address one of the two research questions. Part I concerns the 3D interaction techniques and consists of three chapters. Part II describes the architectures and models for VR applications and 3D interaction and consists of two chapters. Each chapter contains a slightly modified version of a peer-reviewed article published earlier. Figure 1.6 illustrates the thematic division of the two parts and their chapters. To summarize, this thesis presents the following contributions:

- An interaction techniques to enhance the selection of small objects in an active space Virtual Environment. Chapter 2 describes IntenSelect, an improved 3D object selection technique [de Haan 05].

- Integration of familiar 2D widgets in a Virtual Environment. In Chapter 3, we propose *hybrid interface* elements with enhanced selection and manipulation properties for use in a Virtual Environment [de Haan 06].
- An interaction approach to enhance the VR experience for multiple users on a single, projection-based VR system. Chapter 4 describes a technique for multiple-user viewing and interaction [de Haan 07c] on projection-based VR systems.
- A software approach to provide a more flexible adaptation of VR-visualization applications. Chapter 5 discusses a general framework that was developed using *flexible abstraction layers* [de Haan 07b].
- A software approach to better facilitate the design and development of complex interaction techniques and scenarios. Chapter 6 describes the *StateStream* model and architecture, with which one can develop new interaction techniques [de Haan 08a, de Haan 09a]

In the final chapter 7, we reflect on all chapters, discuss the status of the work and give our view on future directions of research.

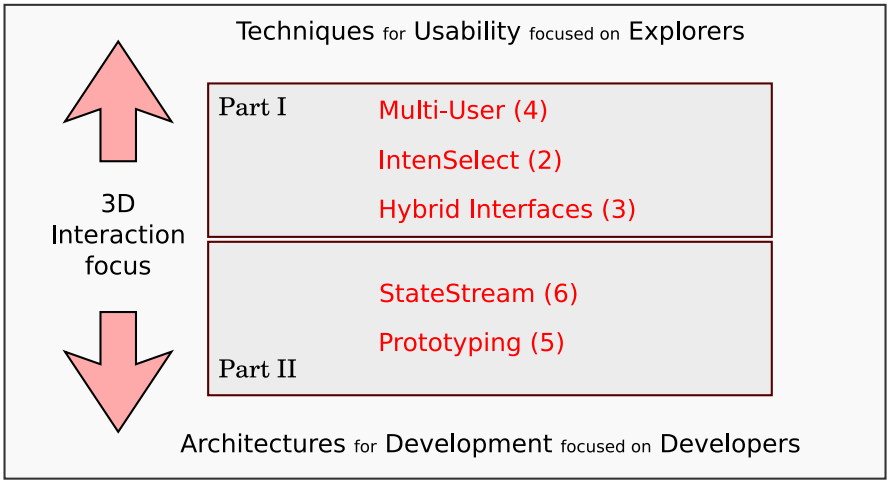


Figure 1.6: Thematic overview illustration of this thesis. The individual chapters (with chapter numbers) are grouped in the two parts (in grey, outlined). Vertical placement indicates the topic focus of usage, ranging from data exploration (top) to development (down).

Part I

3D Interaction Techniques

2

IntenSelect: Using Dynamic Object Rating for Assisting 3D Object Selection

This chapter contains a slightly modified version of an article published earlier. The original peer-reviewed article [de Haan 05] was presented at the Eurographics Workshop on Virtual Environment in Aalborg, Denmark, 2005.

Overview

We present IntenSelect, a novel selection technique that dynamically assists the user in the selection of 3D objects in Virtual Environments. Ray-casting selection is commonly used, although it has limited accuracy and can be problematic in more difficult situations where the intended selection object is occluded or moving. Selection-by-volume techniques, which extend normal ray-casting, provide error tolerance to cope with the limited accuracy. However, these extensions generally are not usable in the more complex selection situations. We have devised a new selection-by-volume technique to create a more flexible selection technique which can be used in these situations. To achieve this, we use a new scoring function to calculate the score of objects, which fall within a user controlled, conic selection volume. By accumulating these scores for the objects, we obtain a dynamic, time-dependent, object ranking. The highest ranking object, or active object, is indicated by bending the otherwise straight selection ray towards it. As the selection ray is effectively snapped to the object, the user can now select the object more easily. Our user tests indicate that

IntenSelect can improve the selection performance over ray-casting, especially in the more difficult cases of small objects. Furthermore, the introduced time-dependent object ranking proves especially useful when objects are moving, occluded and/or cluttered. Our simple scoring scheme can be easily extended for special purpose interaction such as widget or application specific interaction functionality, which creates new possibilities for complex interaction behavior.

2.1 Introduction

Virtual Environments have the potential to increase the insight into complex multi-dimensional models and processes from a wide range of applications. However, VR applications still are not as widespread as their potential would let us expect. The lack of its general acceptance and use is often attributed to the limited flexibility and practical usefulness of the user interface, even though this is considered as an advantage of VR. One of the most basic elements in the complete taxonomy [Bowman 01] of three-dimensional interaction is that of *object selection*. That is, indicating one specific object in the virtual world. When examined very closely, even this basic task of object selection is often surprisingly difficult and can be a source of irritation for the users. We therefore believe that the practical usefulness and acceptance of a wide range of interactive VR applications heavily relies on effective selection techniques.

We mainly employ Virtual Reality techniques as a mean for enhancing the interactive process of scientific visualization and data exploration. In addition to rendering techniques for large datasets, we concentrate on one of the main challenges of scientific visualization in VR as described in the overview by Van Dam et. al.: "... to make interaction comfortable, fast, and effective" [van Dam 00]. Currently our main VR applications focus on the visualization and control of (real-time) simulations of physical processes. We often have to deal with objects dynamically moving through the dataset, such as atoms or glyphs (see Figure 2.1). The researchers involved are interested in the various physical properties, the behavior and context of these objects. To explore these, the object of interest usually first has to be selected.

Nearby objects can be directly selected by manually positioning the spatial interaction device precisely inside its observed graphical representation. This is the so-called *touch* or *direct* selection. For the selection of more distant virtual objects that are beyond arms reach, remote interaction techniques can be employed, of which *ray-casting*, or *laser pointing*, is most commonly used and widely spread. Here, a virtual ray or laser beam emits from the tip of the interaction device, and, by finding an intersection of this ray with an object in the scene, a selection is made. The intuitiveness and low learning curve make the ray-casting technique a very attractive selection technique for many applications. As we also use this ray-casting technique in our applications most of the time, the effectiveness of this selection technique is of great interest to us.

Usually, the techniques described above allow effective and fast interaction with

objects in the scene. In many cases however, we see the difficulties even very skilled users experience in selecting both nearby and remote objects, leading to confusion or frustration. From casual observations we detected that most difficulties are experienced when attempting to select a certain object that is small, thin or remote; occluded or in a cluttered scene; or showing complex behavior such as moving with changing direction and speed. We have devised a new interaction method designed to cope with these three issues in object selection, with the overall goal of assisting the user in selecting the intended object.

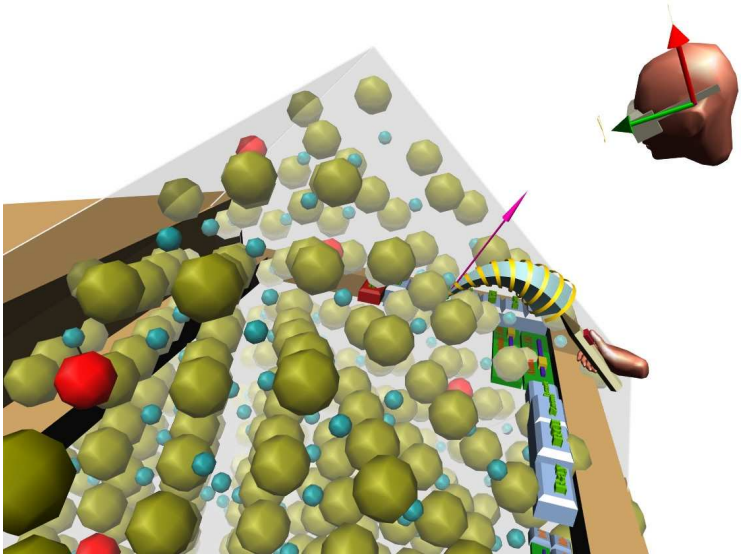


Figure 2.1: Example application: Selecting a moving atom in a heavily cluttered Molecular Dynamics environment.

This chapter is constructed as follows: First, we describe the three aspects of the problem in more detail. Then, we discuss previous work in the area of ray-based and selection-by-volume techniques, as well as other assisting selection techniques for VEs. We continue with a general overview of our algorithm and a more detailed description of our scoring metric. After a description of the implementation of our technique we report on our general impressions on usability. In addition to this, we present the results of an informal user study and compare the performance of our selection technique with other existing techniques. Finally, we discuss our findings and thoughts, and give some pointers for future research and applications.

2.2 Problem Analysis

2.2.1 Selection accuracy: Small and remote objects

The virtual pointer and the attached selection ray are usually controlled by a tracked six-degree of freedom interaction device. The user can position the device to control the origin of direct interaction, while the device rotations control the direction of the ray for remote interaction. For remote interaction the angular control of the virtual pointer dominates over the positional control because angular movements result in amplified movements of an object intersection point at a further distance. In order to accurately pinpoint a distant object using a ray, the angular control needs to be accurate and stable. Although in the last decade tracking quality has increased through improved hardware, filtering and calibration techniques, the interaction accuracy remains negatively influenced by "imperfections" of the human motor functions such as hand jitter and fatigue. Tracker calibration can also be a cause. In non- or poorly calibrated head-tracked environments the misalignment of physical device and virtual selection point hamper intuitive pointing. Even a slight offset can lead to confused or frustrated users, resulting in extra efforts in order to select a virtual object. Often *clutching* is involved, that is, users are repeatedly switching back and forth between navigation, manipulation and selection modes to obtain the desired situation in which they can perform their intended action, such as selecting one specific object. Even if tracking and calibration were perfect, as are in the case of a real physical laser-pointer, it remains a tedious and time consuming task to accurately pinpoint a small object at a large distance. Considering these aspects we believe the previously described discreteness of selection condition should be abandoned for a more error-tolerant approach.

2.2.2 Selection Ambiguity: Occlusion and Cluttering

In many situation where users attempt to indicate or pinpoint something, the object is typically between other objects. In crowded environments there is a high probability that an object is near other objects *(cluttering)*. When pointing at a remote object using ray-casting there is also a high chance that another object is in the path of the ray and triggering an un-intended selection *(selection occlusion)*. By changing the angle and position of the ray or zooming in on the scene, a free line of selection might be found to avoid this occlusion state, but this can be unsuccessful and difficult in a crowded scene. We consider this clutching an unwanted side-effect of the interface, and we believe that a more elegant alternative should be investigated.

2.2.3 Selection Complexity: Moving Objects

In many selection tasks, the previous two issues can be overcome by spending more time and effort to assure a correct selection. However if the scene is not static and

virtual objects or the user show more complex behavior, such as motion, the selection task itself becomes more difficult and exacerbates the previous issues. It is, for example, difficult to accurately pinpoint a small moving target because occlusion and cluttering effects dynamically change. Additionally, there might not be an opportunity to spend more time on the task. Situations where this complex behavior can be seen are, for example, fast moving objects, or objects which are only present for a short period of time as is often the case in real-time simulations or collaborative VR applications. These situations only increase the required effort to correctly select an object, and consequently increase the user's demand for a fast and accurate selection technique. We believe that, in time-dependent situations, we have to provide an interface to cope with the dynamic aspects of the scene.

Both in the real world and in VR it can be difficult to accurately and quickly pinpoint an object, regardless of which selection technique is used. The many degrees of freedom in the interaction allow a myriad of possibilities. The advantage that we have in VR is that the scene-generating software generally has access to many, if not all, aspects of the virtual objects that populate it, as well as all interaction data. When pinpointing in a VE, the degrees of freedom might be strongly reduced by reasoning on this available information, and constraining the selection to only the most likely or intended objects. If possible, the intended selection could be recognized and highlighted to assist the user in the selection task. This *automated assistance* could increase the selection performance over pure manual control, especially in the more complex cases.

2.3 Related Work

Ray-casting selection has been commonly used in all kinds of VR systems, ranging from CAVEs to desktop VR systems [van de Pol 99]. Although this technique provides a way to select remote objects, its practical use is often awkward; classic ray-based interaction has proved to be difficult in selecting objects which are small, occluded and/or moving. As discussed in the previous section, ray-casting requires accurate and stable pinpointing. For a useful selection, this implies that the user can accurately point the virtual ray directly at the object and keep the object selected while performing an action such as pressing a button. A selection to an object can easily be lost as soon as you click the button, nicknamed the Heisenberg effect [Bowman 01].

The spotlight selection technique is one of the first extensions of the classic ray-casting selection technique [Liang 94]. It was presented as a novel selection technique in a desktop-based 3D modeling system with a six-degree of freedom interaction device. To allow a certain level of error in aiming at small and distant objects the mathematical discreteness of the normal ray is "softened". A cone shaped selection volume is attached with its apex to the origin of the co-ordinate system controlled by the interaction device. Objects that fall within this selection volume are potential

candidates for selection and are awarded a score, based on a disambiguation metric. The highest ranking object is highlighted as being the selected object. In the aperture selection technique [Forsberg 96], which is based on spotlight selection, the apex of a conic selection volume is attached to the dominant eye whereas the direction vector of this cone is directed from that eye through the pointer's tip. Attached to the pointer is a circle with fixed radius, which determines the cross-section radius of the cone at the pointer location. By moving the circle towards or away from the eye the user can respectively increase or decrease the size of the selection cone. The eye position and this circle span the cone, and thus interactively determine its size and orientation. For resolving selection ambiguity, a metric similar to the one in the spotlight technique is used. In both techniques the conic selection volume is displayed and the current selection is highlighted. In more recent work, the Improved Virtual Pointer (IVP) is described, using a similar selection-by-volume approach to determine the active object [Steinicke 05]. This technique uses a different scoring metric and uses a flexible ray to visually connect the pointer to the active object.

In order to "automatically" determine the intended object within a selection volume, a balanced metric has to be constructed for an intuitive ranking of the objects. An alternative to using a metric is using additional actions or devices for manual disambiguation, such as twisting the virtual pointer or using a separate button to toggle through selections. Another example is the use of a pop-up dialog containing a list with all the preselected objects [Dang 05]. By selecting an item from this menu, any object within the selection volume can be selected. Although this allows a guaranteed selection of an object, we feel that continuously switching from 3D interaction to menu selection can hamper the user's work flow. The nature of the application and moreover the importance of selection correctness ([Dang 05] describes air-traffic control) will be decisive on the usefulness of this type of interaction.

Another approach uses artificial intelligence techniques to determine the intended object during selection [Wingrave 01]. Here, machine learning is used to detect users' nuances and preferences and to reason with interface information in an attempt to elicit their intentions. The attempts to create intelligent interfaces have shown the feasibility and effectiveness of this approach, but they are currently limited to simple cases and are not yet ready for practical use. In addition, design requirements are presented for such a system, derived from fields such as gesture recognition and automated user interface design. Although we do not use a similar system, it can be of interest to validate the "dynamic" behavior of our algorithm to these requirements.

In our method we attempt to limit both the previously mentioned explicit actions or clutching movements and the use of extra or specialized interaction devices. We limit our technique to only using only the position and the pitch and heading (tilt and azimuth) of a 6 DOF pointing device, equipped with one button for indicating the selection. In this way, our technique is more generic and can be more easily applied to a wide range of VR systems and interaction devices. Furthermore, we

want the selection behavior to be similar to ray selection to reduce the need for extra user training.

2.4 Selection Algorithm

2.4.1 Overview

In this section we will describe our object selection algorithm and the differences between this and existing techniques. Our algorithm consists of the following general stages:

- **Selection volume test**: Determine which objects are inside the conic selection volume.
- **Score contribution**: Objects inside the volume get a scoring contribution, determined by a scoring metric.
- **Score accumulation**: Contributions are accumulated at the objects over time; objects with a score are lowered.
- **User feedback**: The highest ranking object is highlighted and indicated by snapping a bent ray to it.

2.4.2 Selection Volume Test

A cone shaped selection volume is attached with its apex to origin of co-ordinate system controlled by the interaction device. The conic volume is determined by the position and orientation of the pointer's tip, it's orientation and a spread angle. First, we determine which objects are within the range of this volume. For every object in the scene, every frame, we perform an "inside"-test of the object and the conic volume. In our current proof-of-concept we limit our test to the bounding sphere of an object, which is typical for the objects in our user tests. We transform the object's center point position and radius to the co-ordinate system of the pointer. In this co-ordinate system we can easily derive the projection distance on the ray (d_{perp}) and the distance from this projected point to the pointer's position (d_{proj}). Figure 2.2 shows a three-dimensional schematic overview of object P inside the conic volume. Point P lies on a two-dimensional cross section of the cone. This is a circular surface, with the ray defining both its center point and normal, while the radius is determined by the cone spread angle and distance d_{proj}.

$$d_{perp} = \sqrt{x^2 + y^2} \tag{2.1}$$

$$d_{proj} = z \tag{2.2}$$

If the angle α, defined by these two distances, is smaller than the opening angle of our cone (β_{cone}), we consider the object to be inside of the conic volume. For these objects a scoring value will be calculated, while other objects will be ignored in the following stage.

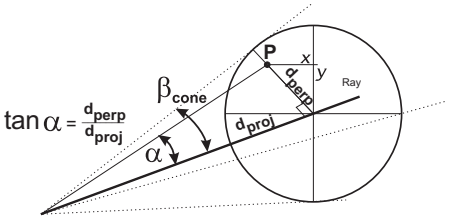


Figure 2.2: Selection volume test: Determining whether point P is inside the conic selection volume.

2.4.3 Score Contribution

The scoring contribution of the objects that are inside the selection volume is determined by a scoring metric. A scoring metric is a formula that assigns a scoring value to an object, based on the properties of the object. As described earlier in Section 2.3, several metrics can be used. For example, in [Liang 94] and [Forsberg 96] an ellipsoidal distance metric is used. Although this metric is designed to be toleratant to aiming inaccuracies, it is still hard to select occluded objects or far away objects when objects are near. In the IVP technique [Steinicke 05], the absolute projection distance similar to (d_{perp}) is used as a metric.

To ensure a seamless and effortless transition from ray-casting selection behavior, we take simple ray-casting as a basis for our scoring metric. To achieve this, we assign the highest score of 1 if pointing is most accurate. That is, the ray is being pointed through the object's center point. Furthermore, we assign the lowest score of 0 if pointing is least accurate, when the object's center point lies on the bounding surface of the conic volume. We achieve this scoring by using the angle α, defined by the distance perpendicular to the ray (d_{perp}) and the distance from this projected point to the pointer's position (d_{proj}), as the main variable of the metric.

$$s_{contrib} = 1 - \frac{\alpha}{\beta_{cone}} \tag{2.3}$$

During testing trials it quickly became obvious that, using this scoring metric, it is easier to select distant objects than it is to select nearby objects. It became clear that the use of this angle α as a direct metric implies larger d_{perp} at larger d_{proj}, and thus tolerates large aiming errors. In more detail, as the d_{perp} is $\sqrt{x^2 + y^2}$, the surface that is inside the cone at larger distance allowed is growing exponentially. Although

this is in essence desired behavior, it feels counter-intuitive. To compensate for this exponential factor, d_{proj} is taken to the power k, a compensation constant:

$$s_{contrib} = 1 - \frac{atan(\frac{d_{perp}}{(d_{proj})^k})}{\beta_{cone}} \tag{2.4}$$

Here we typically choose k to be between $\frac{1}{2}$ (being a linear relation) and 1 (being the original relation). Currently we have chosen k to be $\frac{4}{5}$, based on initial experiments. We acknowledge that this compensation scheme and the compensation factor is dependent on the type of application and user, and can be of interest in further research.

The resulting scoring metric inside the conic volume is shown in Figure 2.3. This figure represents a two-dimensional section of the conic volume with the pointer located in the origin. The black line indicates the ray, while the red dotted lines represent β_{cone}. The gray-scale intensity is used to display the scoring value, white being 1 and gray 0.

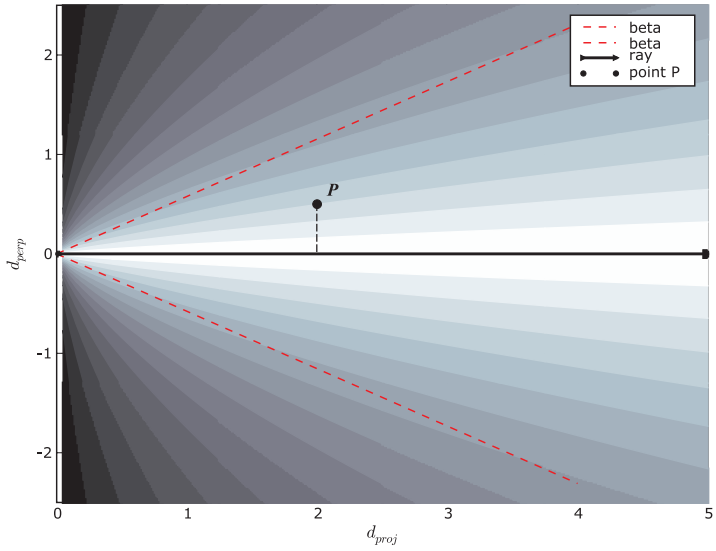


Figure 2.3: Object Scoring Function: Scoring value of point P inside the conic selection volume. Gray-scale intensity represents the scoring value, white being 1 and gray 0. The thick (black) line indicates the ray, the (red) dotted lines represent the opening angle β_{cone}.

2.4.4 Score Accumulation

In previous work the score determines the disambiguation of the selection directly. That is, the position of the pointer and the state of the objects at a point in time (e.g.

a certain frame) directly determine which object is selected. This per-frame static behavior can lead to fast switching between objects when they have similar scores, for example in the case of cluttered or moving objects. Instead of this static behavior we aim at a more dynamic system behavior that can use temporal relation of both objects and interaction. By using this history we hope to create a more meaningful response to the user's actions.

To achieve this temporal effect we don't use the score directly, but the scoring contribution is accumulated over time. Each object maintains its score over several time frames. That is, the score is not reset at every frame, but kept during a sequence of frames. In addition we use a progressive reduction of the scores of all objects, in effect fading out the scores that have been accumulated in the previous frames. The behavior can described by the following formulas:

$$s_{contrib}(t) = 1 - \frac{atan(\frac{d_{perp}(t)}{(d_{proj}(t))^k})}{\beta_{cone}} \tag{2.5}$$

$$s_{total}(t) = s_{total}(t-1)c_s + s_{contrib}(t)c_g \tag{2.6}$$

The contribution score is left unaltered but denotes the score at the current time step (or frame) t. The total accumulated score s_{total} is defined by the total score of the previous time step $(t-1)$ and the current contribution score. Both variables are multiplied by constant scaling factors, c_s and c_g. The constant c_s defines the rate of decay or shrinking of the total score. The constant c_g defines the rate of growth of the score for a selected object. These constants typically define the *stickiness* and *snappiness* of the selection, respectively. If we choose $c_s = 0$ and $c_g = 1$ we get the regular static volume selection without the time-dependent behavior. As these values typically describe a complex trade-off between snappiness and stickiness, their correlation and their influence on the time-dependent selection behavior is not straightforward. Currently, we have chosen these parameters manually based on our initial experiments, balancing both parameters to a comfortable response. In Figure 2.4 the scoring response of a single object in time is shown. Until frame $t = 60$ the object is not inside the selection volume, thus receiving no score. From frame $t = 60$ and later, the object is accurately selected causing the per-frame scoring values to accumulate at the total score. At frame $t = 180$, the object is again outside the selection volume. Here the score is only influenced by the decay defined by c_s. It must be noted that in practical use the selection will gradually enter and leave the conic selection volume, resulting in a more complex scoring response than shown in the figure.

2.4.5 User Feedback

The object that has the highest accumulated score $s_{total}(t)$ at the current frame t is marked as the *intended* or *active* object. If no object has a score higher than zero, no active object is defined. In this case the normal ray is displayed, allowing users to fine-tune their pointing. If an object is marked as the intended object, the end of

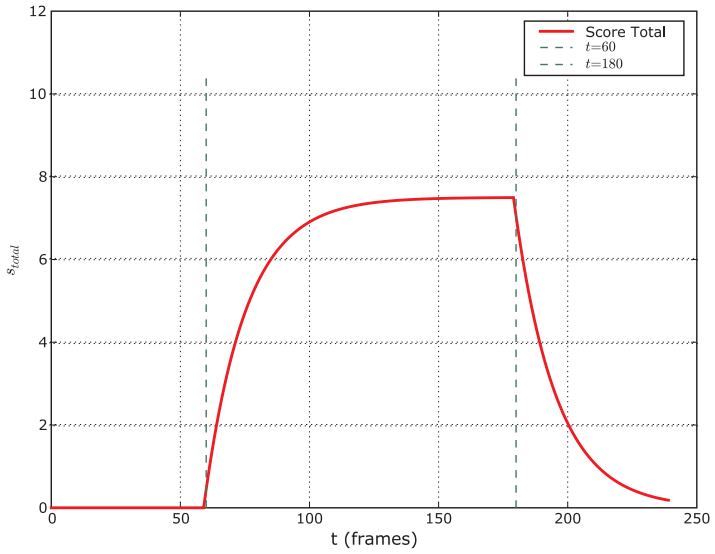


Figure 2.4: Score Accumulation: Time-dependent scoring behavior for a single object.

the selection ray is bent towards it. The bending ray is an extension of the family of our Virtual SpringTools [Koutek 01c], and uses a Bézier curve to determine the geometric shape. The bending is done in such a way that the ray emits from the pointer and the endpoint *snaps* to the object, see Figure 2.5. As long as this object remains the currently activated, the connecting bent ray between it and the virtual pointer is maintained. To achieve this, the shape of the ray is continuously updated to accommodate movements of both pointer and the active object. If another object is activated, the ray will snap immediately to it.

Although we have used the coloring of objects as well as displaying the conic selection volume and several transparent rays deforming towards a number of lower ranking selection objects, we have the impression that these all lead to unnecessary visual clutter and distract the user's attention. Therefore we limit our visual feedback to the bending of the ray and simultaneously show a thin ray that indicates the pointer's aiming direction.

The resulting behavior feels similar to that of ray-casting, only with snapping in less accurate selections. A good effect of the temporal relation can be seen in Figure 2.6. We have recorded a user session (see Section 2.5) where ray-casting is used to select a moving object as accurately as possible. Using this technique, the user frequently *loses* the object. When the same session is replayed and the volume-based selection is used, the object is selected most of the time. Only when the object falls just outside the conic selection volume the selection is lost for a moment. When

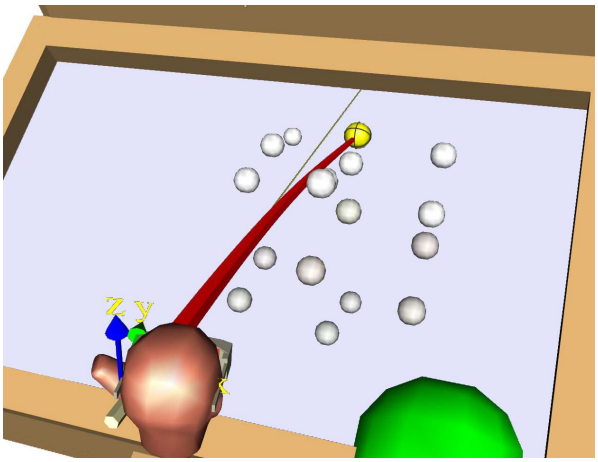


Figure 2.5: IntenSelect in action: The user is selecting an occluded, moving and far-away object. A thin line indicates the pointing direction, while the bending ray remains snapped to the highest ranking object.

replaying the session with IntenSelect, the temporal filtering maintains the selection the entire movement.

We must note that many of the parameters and techniques shown here can have many variations and can be optimized in various ways to adjust behavior and maybe improve performance. In our current applications we have not noticed significant performance penalties. Nevertheless, if performance issues would arise in more complex scenes, various optimization techniques can be employed.

2.4.6 Flexibility and Extensibility

In addition to the *default scoring metric* for simple virtual objects, we have the possibility to assign *special-purpose scoring functions* to customize the behavior of the selection interface. By using user-defined object callbacks in the scoring function we can enhance or redefine specific scoring distributions resulting in different selection behavior. This specialized selection behavior can be customized for special virtual objects or on a per-application basis, allowing application developers to tune the interaction performance for specific purposes. In this way we can facilitate improved selection behavior of items of special interest such as buttons and sliders in the user interface. Another example would be the selection of a door in an architectural model. The door itself could redirect its scoring contribution to the door handle (which itself is more difficult to ray-cast), directing the selection and the user's attention to the more interesting or functional virtual objects.

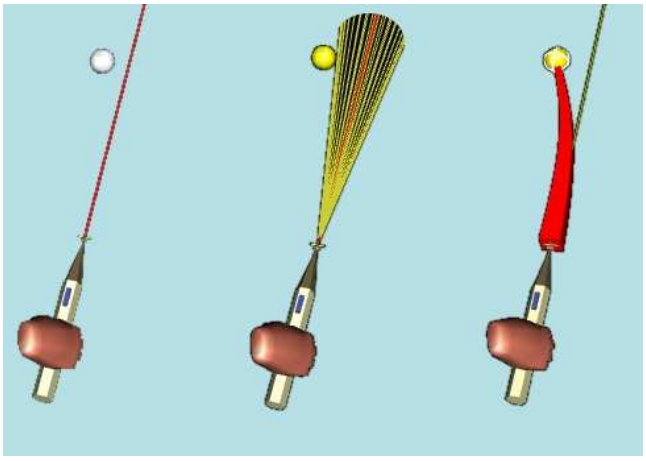


Figure 2.6: Selecting a moving object: Comparing a frame from a recorded session of (a) ray-casting (b) volume selection, with visible conic selection volume, and (c) IntenSelect. Moving objects can easily fall just outside the selection volume for a short period of time.

2.5 Implementation

We run our applications on our Responsive Workbench (RWB), a tabletop projection-based VR system. We use dual DLP projectors to display a stereoscopic image of 1400 x 860 pixels at the glass plate with 180 x 110 cm display dimensions. A Polhemus Fastrak electromagnetic tracking system is used to track our interaction devices.

We use our RWB-Library [Koutek 03], a custom VR software library built on top of the OpenGL Performer scene graph API. The library provides basic interaction functions for selection and manipulation and contains a set of 3D widgets such as buttons and sliders, and handles all necessary tracking and display functionality. The library uses a default RWB-interactor class to allow selection and manipulation of these RWB-objects directly with the stylus or by using ray-casting. 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 state machine, driven by interaction events.

The IntenSelect technique is currently implemented as a special RWB-interactor class, and allows easy integration with existing RWB-applications. Instead of checking for bounding volume or ray-casting hits, the interactor determines the ranking of RWB-objects in the scene at every frame. If a new highest ranking object is found, its touch callback function is called. The previously winning object is deselected by calling its un-touch callback. The selection ray is bent to snap to the center of the bounding volume of the highest ranking RWB-object. If the user presses the stylus button, the object is selected and the pick callback is called. As long as the button is

pressed, the object ranking is not updated. The interactor will be in the manipulation state. To maintain the connection to the object being manipulated, the bending ray is being continuously updated, as described in Section 2.4.5.

2.6 User study

To verify our technique we have conducted a small informal user study. In this setup we compare IntenSelect with regular ray-casting technique and our volume-based selection without temporal accumulation. We use the same display and interaction hardware without re-calibration between tests. Furthermore we have fixed the frame rate to 60Hz, leading to a timing resolution of 16.67 ms. After a small introduction to the VR system, the user starts out with a 2-minute training session to get acquainted with the interaction styles and the display environment. This is followed by a series of small tests in which selection time is measured.

2.6.1 Test setup

In our test scene the user can use the selection technique to select single objects in the scene. A test scene consists of a pre-generated set of simple virtual objects which are both similarly shaped and colored. First, the subject holds the pointer inside the *starting sphere*, which is a large object located at a comfortable position on the right-hand side of the scene, see Figure 2.7. Once this starting sphere is selected, one of the objects from the set is highlighted by a distinctive color. As long as the stylus is inside the starting sphere, no ray information is displayed nor the task time has started. We instruct the user to first find the object and only then initiate the selection. In this way we exclude the finding activity of the subject from the timing, and prevent coincidental hidden objects. As soon as the user has visually located the highlighted object, the pointer can be moved from the starting box. At this moment, the used selection technique is activated and the timer starts ticking. As soon as the user selects the highlighted object by pointing at it and pressing the stylus button, the timer is stopped. The user is then instructed to return the pointer into the starting box. Now, another object is highlighted and the sequence starts from the beginning until the test is complete.

We have created two scenes, one with small static objects and one with larger moving objects. The static scene is created to test only the selection performance of small, distant objects, of which some problems were described in Section 2.2.1. It consists of an array of 8x8 small spheres at a distance of approximately two meters inside the workbench. The moving scene consists of 16 larger spheres which quickly follow a predefined periodic path through the scene. During these movements, the objects continuously intersect, clutter and occlude each other. The moving scene tries to mimic real applications with complex moving objects and occlusion, see Sections 2.2.2 and 2.2.3.

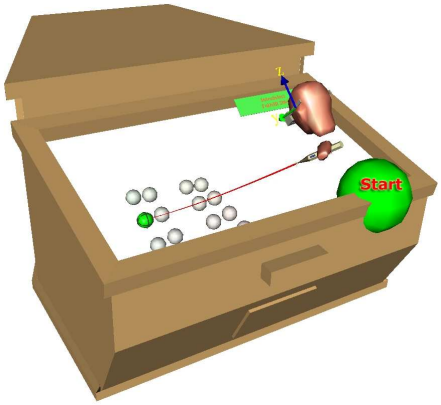


Figure 2.7: User Test Scene: Using IntenSelect to select a moving object amongst other moving objects. The spheres make fast periodic movements during which they continuosly intersect, clutter and occlude each other.

The generated scenes are created off-line and are accompanied by a predefined sequence of objects that are to be selected. Each subject uses the three different selection techniques, ray-casting, selection-by-volume and IntenSelect, on the same selection sequence in both scenes. This leads to a total of 6 small tests per user, each consisting of 10 object selections. We start out with the three interaction techniques on the static scene, followed by the same techniques on the dynamic scene. To avoid some of the training effects in favor of our technique, we start out with the IntenSelect technique, followed by the volume selection and finally the ray-casting technique. If these training effects do occur, they mainly work in favor of the regular ray-casting and volume-based selection most. Nevertheless, users might also reduce their accuracy in pointing in the first trials, reducing the performance in ray-casting.

2.6.2 Test Results

A total of eight subjects have successfully completed all of the six tests, resulting in 480 time values. The test group mainly consisted of male colleagues in the age range of 25 to 30, of which only three had hands-on experience with interaction on Virtual Reality. In Figures 2.8 and 2.9 the quantitative results of the tests are presented. In Figure 2.8, the leftmost three boxes indicate the selection speed in the static scene, while the remaining three indicate the selection speed in the dynamic scene. The y-axis indicates the object selection time, measured in seconds starting from the leaving of the starting sphere to the selection of the object with the pointer. For each box, eight samples are used, each representing the user's averaged object selection time for that session. Each box extends from the lower to the upper quartile value of the

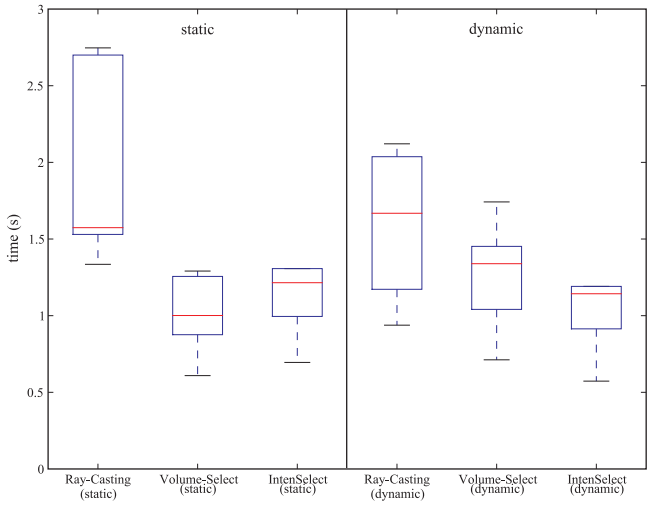


Figure 2.8: Test Results: Box plot of averaged object selection times for the three techniques in static scene (left) and dynamic scene (right).

samples, while the median value is indicated by the middle (red) line in the box.

In Figure 2.9, a relative comparison is made between median values of the three selection techniques. Here we take the ray-casting technique as the reference technique, and express the performance of the other techniques as a ratio of this reference. First, we average all selection times of all users per session, leaving out the best and worst timing value of each user. Then, we normalize these average timings with respect to ray-casting to obtain relative timing values, and take their inverse to obtain relative speed-ups.

First, it becomes clear that, in both scenes, generally, ray-casting is out-performed by the two improved selection techniques. All users appreciated the helpful assistance in pointing and found the improved selection methods a relief in contrast to regular ray-casting.

In the static scene, somewhat to our surprise, the IntenSelect technique with temporal accumulation does not seem to perform better than the volume-based selection technique. The more experienced users reported that, in the case of IntenSelect, the ray felt too sticky. This forced them to wait a small amount of time for the ray to update its selection to the intended object. The less skilled users however did not report any noticeable differences in the two latter techniques, but nevertheless they performed best with IntenSelect. We believe that their generally slower interactions allow the temporal scoring to update quickly enough, effectively showing similar behavior to the normal volume selection. In addition, their inaccurate and jittery aiming probably led to more incorrect selections using the basic volume selection.

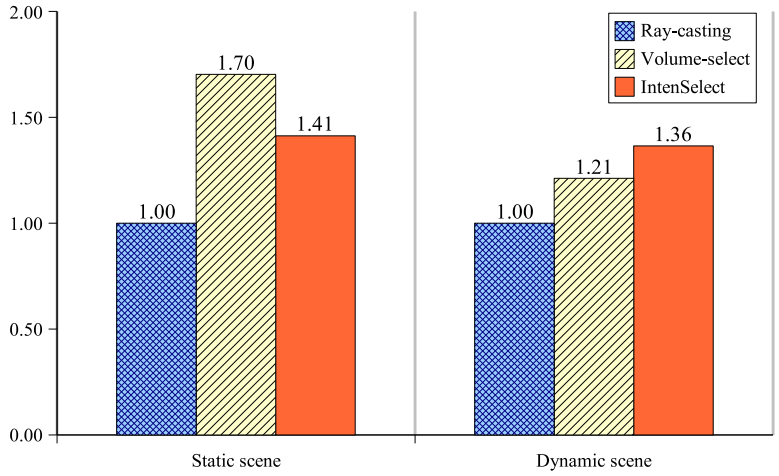


Figure 2.9: Test Results: Relative average selection speed-up of volume-selection and IntenSelect compared to the ray-casting technique.

Furthermore, we believe that if cluttering increases (effectively decreasing error tolerance), the selection with only volume-selection technique becomes more difficult.

In the dynamic scene the IntenSelect was found the most effective technique. The continuous occlusion and cluttering effects hamper the use of the normal volume-selection, and lead to many wrong selections. Individual results show that, in only in one or two cases, very experienced users performed best using simple ray-casting. These users reported that they had gained experience from the previous sessions. Typically, they had learned to wait for the right moment in the periodic movement of the objects.

Although our initial tests show an increase in average performance, it is too early to make decisive quantitative statements on this. A larger and more formal test setup is required to achieve a more reliable results and to perform statistical evaluation. For a more reliable test and an elimination of any unwanted learning effects, we should create a better experimental design. For example, we could increase the number of users (e.g. n=25), randomize the trials and use both between and within-user experiments. Furthermore, we need to gain more insight in the complex influence of the individual object and scene characteristics on the performance of our technique.

2.7 Discussion and Future Work

In this chapter we have discussed the design IntenSelect, an improved ray-selection technique for object selection in VR. We have tested this technique in comparison

with others, on the Responsive Workbench. Early user experiments may indicate that our technique appears more effective in selecting small objects, especially when these are moving and when clutter or occlusion effects occur. Moreover, all users appreciated the helpful assistance in pointing. Before a more definite statement can be made though on the value of this technique, a number of factors will have to be considered. First, we have used a simplified testing set of virtual objects, consisting of only spheres. Second, we have to compare the technique to other, also non ray-casting based selection techniques. Finally we have to extend our current user test to a larger set of test-subjects and more narrow tasks to gain more detailed insight in the various parameters, their correlation and their impact on user-performance. In addition, we will have to evaluate our technique on a wider range of systems with various display and tracking hardware, including our PowerWall, desktop VR systems such as the Personal Space Station [Mulder 02], and the CAVE.

As we briefly described in our introduction, our main VR applications focus on the domain of data visualization and control of physical simulations. Our next goal is to apply the IntenSelect technique in these applications. We plan to enhance general application control by applying a special scoring function for 3D widgets such as buttons. In addition, we will apply the IntenSelect technique and required application specific optimizations to the selection of application objects, such as atoms or glyphs.

We have observed the powerful effect of *hinting* gestures when users follow a moving object or try to disambiguate between two remote, adjacent objects. The temporal effects of the scoring function allow these more advanced, time-dependent user interactions. The flexibility of the various scoring functions and their intricate time-dependent relations open up a myriad of opportunities for specialized interaction behavior.

3
Hybrid Interfaces in VEs: Intent and Interaction

This chapter contains a slightly modified version of an article published earlier. The original peer-reviewed article [de Haan 06] was presented at the Eurographics Workshop on Virtual Environment in Lissbon, Portugal, 2006.

Overview

Hybrid user interfaces (UIs) integrate well-known 2D user interface elements into the 3D virtual environment, and provide a familiar and portable interface across a variety of VR systems. However, their usability is often reduced by accuracy and speed, caused by inaccuracies in tracking and a lack of constraints and feedback. To ease these difficulties often large widgets and bulky interface elements must be used, which, at the same time, limit the size of the 3D workspace and restrict the space where other supplemental 2D information can be displayed. In this chapter, we present two developments addressing this problem: supportive user interaction and a new implementation of a hybrid interface. First, we describe a small set of tightly integrated 2D windows we developed with the goal of providing increased flexibility in the UI and reducing UI clutter. Next we present extensions to our supportive selection technique, IntenSelect. To better cope with a variety of VR and UI tasks, we extended the selection assistance technique to include direct selection, spring-based manipulation, and specialized snapping behavior. Finally, we relate

how the effective integration of these two developments eases some of the UI restrictions and produces a more comfortable VR experience.

3.1 Introduction and Motivation

Poor user interfaces (UIs) in virtual environments have long been cited as one of the major factors preventing widespread acceptance of Virtual Reality. A significant amount of research has been devoted to developing new and improved VR UIs, but a consensus has not yet been reached on what a good VR UI is. One trend has been to focus on using so-called hybrid UIs, which incorporate well-known 2D user interface elements into the 3D environment, rather than developing entirely 3D interfaces using new metaphors. These interfaces benefit from the familiarity and their relative portability across a variety of VR systems. However, tracking inaccuracies and limitations on rendering text in VR generally necessitate large and crude widgets. Large widgets make for bulky user interfaces, which can occlude objects in the scene. They also limit the size of the 3D workspace and restrict where other supplemental 2D information can be displayed.

We primarily employ Virtual Reality as a means for enhancing the interactive scientific visualization and data exploration process. Our applications focus on the visualization and control of (real-time) simulations of physical processes, and Cloud Explorer [Griffith 05], which is illustrated in Figure 3.1, is currently our main application of interest. The goal of this application is to facilitate cumulus cloud life-cycle studies. Large data sets result from atmospheric simulations, from which various information modalities and features are extracted in an off-line preprocessing phase. The resulting data can be interactively explored in a virtual environment, such as our Virtual Workbench, which is equipped with tracked glasses, stylus and a transparent acrylic hand-held panel called the PlexiPad.

During the course of developing this and other applications, we increasingly came across situations that warranted 2D input, 2D output, or both. Our existing UI solutions, namely primitive buttons, sliders, and graphs, were not able to meet all of our requirements. They were promising, but they suffered from the common problems of being overly large and inflexible. In addition, they often remained frustrating to interact with, even after improving tracker registration and the use of passive haptic feedback on the Workbench surface or the PlexiPad. To address these issues, we developed two strategies in parallel: an improved interaction technique and an improved hybrid interface, which we then successfully integrated.

The remainder of this chapter is organized as follows: first we describe some previous work in the field of hybrid interfaces and interaction assistance. In Section 3.3 we describe our work on the hybrid interfaces, followed by the extensions of the interaction technique in Section 3.4. Finally, we describe the combination of these developments and discuss our results and future work.

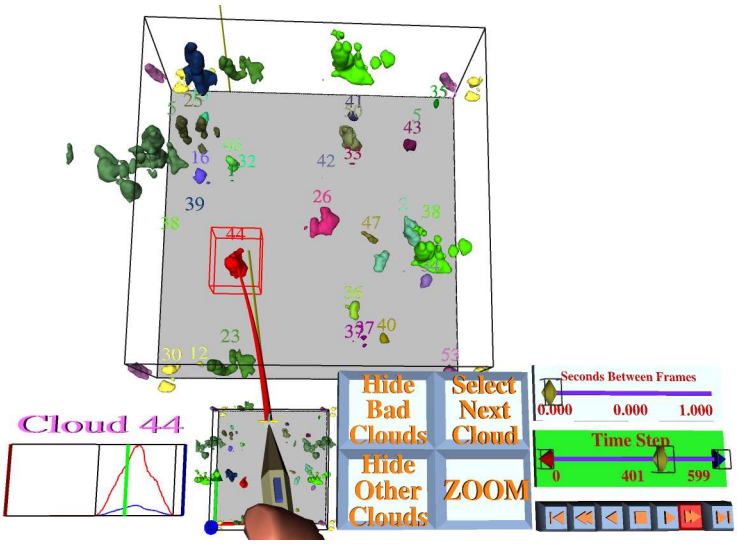


Figure 3.1: Overview of the original Cloud Explorer, our application for cumulus cloud life-cycle studies

3.2 Related work

Windows and window-like constructs have appeared in VEs for many years, and more examples appear in the literature than are listed here. One early example is from Fisher et al. [Fisher 86], where information windows and reconfigurable control panels are briefly mentioned. Other early work focused on importing X Windows into VEs through the use of bitmaps [Feiner 93], textured polygons [Dykstra 94], and specialized widget toolkits [Angus 95a, Angus 95b]. This approach, however, favors the use of a suite of existing 2D applications, which need only minimal communication with the VE application, e.g. receiving mouse events. Another popular window metaphor has been the hand-held window. These have been implemented for augmented reality [Szalavári 97], head-mounted displays [Angus 95a, Bowman 98, Lindeman 99a], workbenches [Schmalstieg 99, de Haan 02], and using a 3Com PalmPilot in a CAVE-like environment [Watsen 99]. This approach is tightly integrated into the VE, and typically uses a pen-and-tablet metaphor for user input. The window is fixed to the panel, though, and it limits the user to one active window at a time. Cuppens et al. [Cuppens 04] and Larimer and Bowman [Larimer 03] have made two recent attempts at more complete 2D UI solutions for VEs. Cuppens et al. specify their UI via XML and provide interaction via a PHANToM device for a pen-and-tablet metaphor. They currently limit the UI to menus, toolbars, and dialogs. Larimer and Bowman focus on CAVE-like environments by placing their windows tangent to a user-centered sphere. Their windowing library builds on an existing

GUI toolkit, and it relies on ray-casting for interaction.

Various approaches are used to alleviate the difficulties in interaction with (small, 2D) interface elements in a VE. The use of (passive) haptic feedback and constraints limits the users actions to match 3D location of the elements. The use of physical surfaces such as the projection screen of a workbench or a tracked handheld panel [Szalavári 97] are examples of placeholders which provide passive feedback. User tests by Lindeman et al. [Lindeman 99b] indicate that the addition of passive-haptic feedback for use in precise UI manipulation tasks can significantly increase user performance, and that users prefer this type of operation. In [Frees 05, Osawa 05] the Control-Display ratio of the interaction is changed dynamically, and user's movements are scaled down to support small and precise interaction. As this approach affects the co-location of the interaction device in the virtual environments this solution is mainly limited to non see-through HMDs.

Selection-by-volume techniques are used for selecting small objects, which are difficult to select using regular direct- or ray-based selection [Dang 05, Liang 94]. There are, in general, two approaches for singling out an object between many (cluttered) objects in a selection volume, which is usually the case in grouped user interface elements. First, the distinction can be made explicitly by the user, for example by using a menu [Dang 05] or a selection sweep to single out one item [Steed 04]. These require a switch of interaction mode (or extra buttons), which is not desirable for menu or widget operation. The other approach is the use of scoring metrics to automate the determination of the best scoring object [Steinicke 05, Liang 94]. As these metrics are calculated on a per-frame basis, they are hampered by the same tracking inaccuracies and hand jitter that make regular interaction techniques so difficult. In the IntenSelect technique these per-frame scores are accumulated over time for each object, see Chapter 2 or [de Haan 05]. As a result a dynamic, time-dependent object ranking with history is constructed. The highest ranking object is indicated by snapping the bending selection ray to it.

3.3 Windows and Widgets

Windows in virtual environments can serve as tools for both input and output. Dialog boxes and tool bars are common examples of input windows, and they have been implemented in VE applications such as [Coninx 97, Teylingen 97]. Two examples of output windows in VEs include a map of an airplane interior [Angus 95a] and a simple clock [Larimer 03]. 2D input functionality is often used for system control, such as enabling or disabling modes in the VE, and symbolic input, such as entering numbers. Virtual environments incorporating 2D input elements are termed hybrid interfaces. 2D output, on the other hand, is used to provide additional relevant information about the VE. If done properly, this helps users learn more about the VE [Bowman 99]. Virtual environments with such information are called information rich.

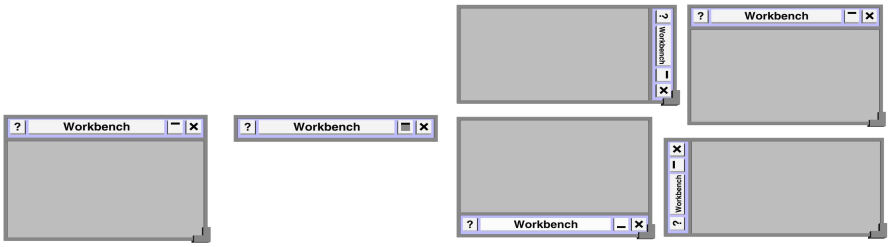


Figure 3.2: A window with information, title, windowshade, and hide widgets in the titlebar and a sizing widget in the lower right-hand corner (left). The same window can be used in windowshade mode (middle) and have four possible title bar locations (right).

When developing our hybrid interface, we tried to meet several goals. We want our interface to be tightly integrated into the VE to facilitate bi-directional communication, thus eliminating the barrier between "information rich" and "hybrid interface". Our primary VR setup is a Virtual Workbench so our solution must work well with it. The UI elements should not be unreasonably large because are intended to be supplemental rather than the primary focus. The interface should be intuitive to use. There should not be a limit on the number of windows visible, and the user should have control over where and which ones are visible. The windows and widgets should be natively rendered in 3D for aesthetics, readability, and to take advantage of the suggested increase in accuracy provided [Lindeman 01].

In the remainder of this section, we present the specifics of our implementation. We describe the windows, the modifications to the existing widgets, and the relevant motivations behind the decisions we made.

3.3.1 Windows

Figure 3.2 illustrates the basic window we came up with. It features a title bar with familiar widgets, a body area, and a sizing widget. The window is rendered as geometry with the window body being flat, while the border and widgets are 3D. This gives better visual cues when the button widgets are pressed, and it allows for direct selection of the widgets by placing the stylus tip inside of them. The window has its own 3D coordinate system to allow for 3D content to be attached to it or placed on it, and it takes advantage of the stencil buffer to allow 2D content to be overlayed onto it. As the window is resized, the components must be moved and resized rather than rescaled to avoid awkward stretching effects.

We designed the windows to be fixed to a particular plane in space. Within the plane, they can be moved and resized, depending on the window, as if it were on a 2D desktop. Translation perpendicular to the plane or rotation is disallowed to preserve the likeness to traditional 2D interfaces. For our purposes, we fix windows onto two planes: the surface of the Workbench and the plane of our PlexiPad. The

Workbench surface has the advantage of being the focus plane while the PlexiPad can be moved out of sight by the user when it is not needed. Furthermore, they both provide optional passive haptic feedback, which has been shown to be both intuitive to learn [Bowman 98] and to speed certain 2D tasks [Lindeman 99a]. In other systems, other planes may make more sense, such as the walls in a CAVE-like or Power Wall system.

The window in Figure 3.2 (left) has a title bar, four title bar widgets, and a sizing widget. However, they are all optional, and it need not have any of these widgets if they are not necessary. The sizing widget, as the name suggests, is used to resize the window. The remaining title bar widgets are the information widget, the title widget, the widget, and the hide widget. The information widget displays extra information about the window. The title widget shows a caption for the window, can be used to move the window around, or both. The windowshade widget hides the body of the window so only the title bar is visible, and then shows the body again if used twice, see Figure 3.2 (middle). The hide widget hides the window. We have made it possible to place the title bar on any side of the window, see Figure 3.2 (right). This has two advantages. First, in systems such as the Workbench, the user is closer to the bottom of the window. With the title bar at the bottom, it is not obscured by any potential 3D content in the window, and it is closer at hand if the user wishes to move the window. Secondly, when the window is in windowshade mode, the title bar determines where the window "pops out". This allows for windows to be arranged at the edges of the Workbench or the PlexiPad in a convenient manner, and only restored when their functionality is needed.

3.3.2 Widgets

Our windows make use of both our set of existing 3D widgets, and a new set of widgets that are more window aware. We have also extended some of our existing widgets to take advantage of the windows. Figure 3.3 illustrates a window with a button from our old widget set, a slider from the new widget set, and a tooltip. To use the existing widgets, a developer need only place them in the window's coordinate system at an appropriate location. The new widgets may only be placed on windows or in their title bars because they occasionally inform the window about their state.

One of the important new widgets is the tooltip. We have included it because it helps address the problem of VE clutter. Text must be reasonably large to be legible in a virtual environment, and it is often a challenge to think of a descriptive caption for menu items or buttons that is not too long. At the same time, experienced users will not have much need for overly descriptive captions as they are already familiar with the commands available to them. With tooltips, smaller buttons, sometimes with an icon instead of text, can be used thereby saving both space and not leaving the novice user stranded. Tooltips can be easily associated with all of the new widgets, and the older widgets have been extended to provide them as well, if deemed necessary. Figure 3.3(left) illustrates the tooltip for a button being shown.

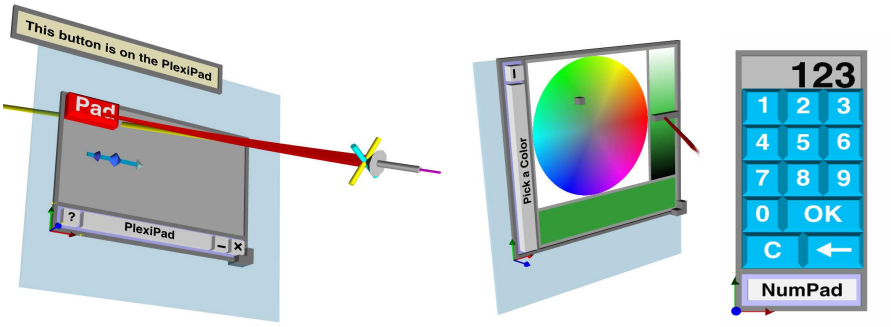


Figure 3.3: A window with a button, a slider, a visible tooltip, and the standard window widgets (left), a color picker dialog (middle) and a numeric entry dialog (right).

3.3.3 Dialogs

In 2D UIs, dialogs or dialog boxes are commonly used to request input from the user. A familiar example is the dialog box to open a file. Dialog boxes have already found a home in virtual environments as well. Both [Larimer 03] and [Cuppens 04] incorporate dialog boxes for the purpose of influencing the environment. We have constructed two simple dialog boxes, which are illustrated in Figure 3.3 (middle, right).

The first is a simple color picking dialog box. It uses the HSL color model to allow the user to interactively select a color. She does so by separately moving a widget to pick the hue and saturation and a widget to pick the lightness. As she manipulates the widgets, they are constrained to the appropriate places on the window, and the rectangle at the bottom of the window updates to show the currently selected color. We draw the color wheel and lightness bar with fragment shaders to give the user the freedom to make the dialog as large or as small as she prefers.

The second dialog is a simple numeric entry dialog. When placed on a device such as the PlexiPad, the user has ready access to it only at those times he feels he needs it. In Cloud Explorer, this dialog is used to allow the user to select a particular cloud by its identifying number.

3.3.4 The Graph Window

The new graph window from Cloud Explorer (Figure 3.4) is an example of a window that "puts it all together". Whenever a cloud in the data set is selected, a graph window is shown with information about that cloud. The window has many of the normal widgets, but it also has two extra title bar widgets. These widgets add or remove graphs from the window. A number of plots can be selected to display in each graph. As Cloud Explorer runs, the user browses through time in the data set,

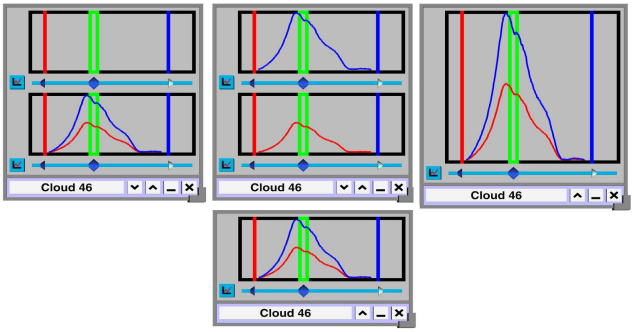


Figure 3.4: The graph window features bi-directional communication and extra title bar widgets. Graphs are added or removed by picking the up or down widgets in the title bar. The plots shown in each graph are selected from a dialog which pops up when the button widget to the lower left of each graph is picked.

and the slider updates to reflect the current simulation time, as well as the limits of the playback. Furthermore, the user can directly adjust both limits and the current time step from the graph window.

3.4 Supporting Interaction in VR

Results of our previous user test indicated that selection-by-volume techniques and IntenSelect can improve object selection performance, especially in difficult situations where objects were small, moving and cluttered [de Haan 05]. Although the selection technique was originally designed with the dynamic nature of data objects such as used in Cloud Explorer in mind, some of its properties proved to be very useful in the fast control of small and cluttered user interface elements. The scoring mechanism allowed easy integration of more advanced features, which we have used to support more elaborate interaction tasks in real VR applications, specifically in the case of interface control.

3.4.1 Transition between direct and remote interaction

In near-field VR systems, such as our Workbench, often a mix of direct and remote interaction techniques are used. Remote interaction techniques are used to interact with objects that are out of (comfortable) reach of the user's arm and interaction device. For example, virtual objects that are placed under the glass surface or near the screen edges are not easily selected with direct techniques. However, direct interaction techniques in particular benefit from co-located interaction and passive haptic feedback from the PlexiPad and the Workbench surface. Direct interaction is never-

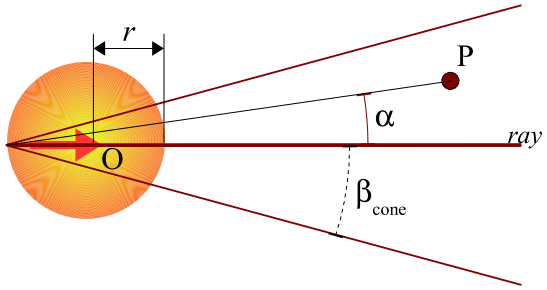


Figure 3.5: Scoring Metric: Selection sphere combined with the selection cone (2D section)

theless also hampered by hand tremor, tracking inaccuracies and calibration errors. To combine the benefits of direct interaction and supportive selection we have extended our IntenSelect technique to support direct interaction. We a use selection-by-volume approach to provide a fuzzy preselection of the objects that might be of interest. In the case of remote interaction we use a cone-shaped volume to accomplish this. For direct selection, we use a sphere surrounding the tip of the interaction device. In this selection sphere we also apply a scoring metric to assign scores to objects. The combination of the scoring metrics result in a single ranking list based on accumulated score. In this way, both remote and direct interaction can be performed without ever switching the interaction tool. As shown in Figure 3.5, both scoring metrics are merged to allow a seamless transition of the scores. We have emphasized the direct scoring value, which results in nearby (directly selected) objects being preferred by the selection mechanism. In Figure 3.6, we show the visible selection volumes of both the direct and remote scoring. The radius r and cone opening angle β are determined beforehand, based on the system configuration regarding tracking accuracy and size of the VE.

3.4.2 Snapping behavior

When an active object is highlighted by the use of our selection technique, the ray bends and snaps to a certain point on the object. For scenarios where only selection on a object level is relevant, this snapping point is only useful for visual feedback. In our previous implementation, the snapping point of an object was defined as the origin of the bounding volume of an object, which is usually the center point of the object. In some cases however, other snapping points must be provided. One of the more important cases where such a snapping point is needed, is object manipulation. In that case, the snapping point defines an origin of interaction around which the object is transformed. We will discuss the manipulation in the following section. Another scenario is the use of an exact snapping point in information panels or controls, where the snapping point on a surface triggers an information query (e.g. color

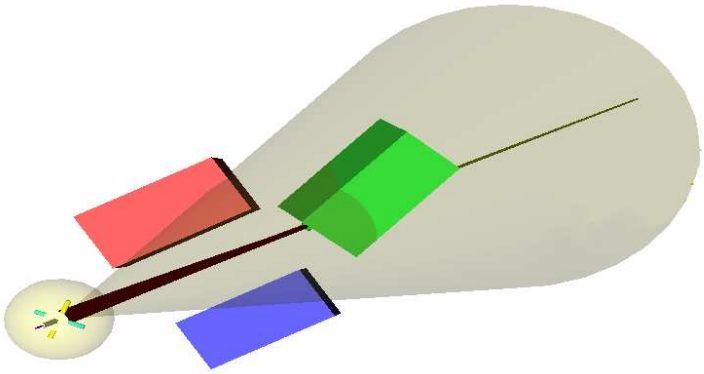


Figure 3.6: Selection Volumes: The selection sphere (direct interaction) combined with the selection cone (remote interaction)

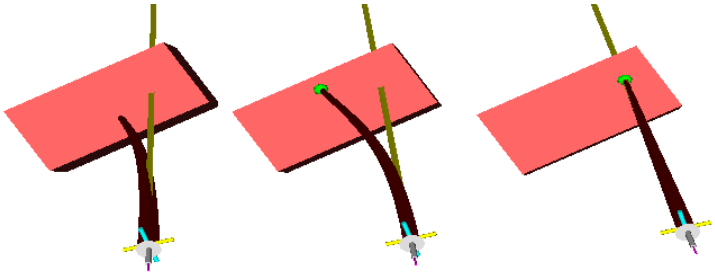


Figure 3.7: Snapping Point variations, from left to right: Center point, (pre-defined) fixed point, and ray-based intersection point

picking dialog).

In our current implementation we now also provide user-defined and ray-intersection snapping points for use in interaction tools and widgets. When ray-based snapping mode is defined for an object and ranks highest on the scoring list, a regular ray intersection test is performed on the polygons of the active object. If an intersection is found, this point is used as the snapping point (similar to ray-casting). If no intersection is found, either a fixed or the last available snapping point can be used. Figure 3.7 depicts the three variations in snapping modes. Although still in early development, we have also experimented with nearest intersection point estimation as an optimal snapping point.

3.4.3 Object Manipulation

We have extended the basic IntenSelect selection technique with a manipulation technique to allow straightforward object manipulations with continuous visual feedback. By pushing the stylus button, the user seamlessly activates the manipulation mode of the currently selected object. As long as the user holds down the button, the bending ray will remain snapped to the active object. For default objects, the manipulation mode enables the repositioning and reorienting of the object. Before starting such manipulation, the original spatial transformation between the virtual pointer and the active object's snapping point are stored. The subsequent translations and rotations of the stylus are then transferred to the object, similar to ray-based manipulation and the Spring Tools [Koutek 01c]. If the object can perform these transformations unrestricted, the original bending of the ray will be maintained during manipulation.

If, however, the object's transformations are influenced or restricted by interactions with the environment, the original transformation cannot be (fully) performed. Examples of these transformation restrictions include implicit object movement, constraints or collisions. In these cases, the resulting object transformation under all influences is applied, and the ray is deformed to match the object's final pose. As a result, the bending of the ray will maintain the connection to the object, regardless of external restrictions. In this way, we provide continuous visual feedback to the selection and manipulation actions. In Figure 3.8, the manipulation sequence of an object is shown. During this manipulation, a collision prevents the object from further motion, while the ray is deformed and remains connected to the snapping point on the object.

3.4.4 Scoring Response control

The IntenSelect scoring mechanism generally applies the same scoring metric to all the objects in the scene, regardless of their size, orientation or movement. That is, the same score determination is used per frame, including the same scaling parameters, the stickiness and snappiness, used to describe the time dependent behavior of the accumulated score. In our previous user test, some skilled users commented on the imbalance between these scaling parameters. They found that, for easy selection tasks in non-occluded, non-cluttered situations, a higher stickiness was hampering fast interaction. This effect was also noticeable in some elements of our test results: in the simplest selection task the time-dependent scoring technique was often outperformed by the snappier, time-independent version. As a contrast, the time-dependent scoring was preferred in selection tasks in tests where objects were moving and cluttered.

We take advantage of this observation by introducing specialized, per object, scoring behavior. This allows us to specify custom scoring response parameters for those objects which might benefit from this. For example, small and cluttered ob-

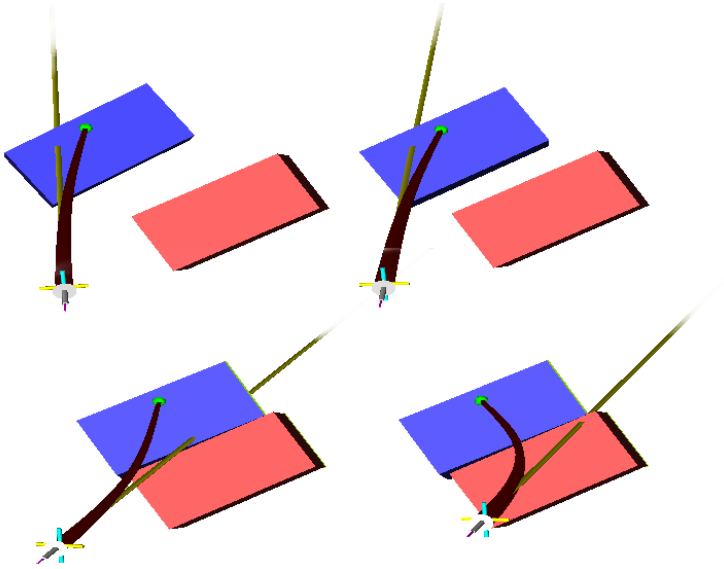


Figure 3.8: Manipulating an object using the flexible ray. The blue (left) object is unable to move through the red (right) one, and so the ray must be flexible to remain connected to the blue object.

jects can be appointed higher stickiness to ease their selection and disambiguation. During local and precise interaction in this cluttered region the user generally slows down to select the intended object. However in regular interaction situations, tracking and hand jitter will make the necessary disambiguation between the small objects difficult. By increasing the stickiness of the objects, we effectively smooth their scoring values over time to compensate for this. The scoring mechanism which controls the snapping behaves as a low-pass filter. As a result, the bending ray will, albeit with a certain delay, snap to the highest ranking object and will remain snapped for certain longer period of time while the user is trying to hold the selection. The delayed selection caused by the high stickiness can provide the user sufficient opportunity to trigger the intended object. To illustrate this, Figure 3.9 shows the different accumulated scoring responses when a conic volume sweeps three adjacent objects at a constant speed. For the higher stickiness setting, the score is more spread out, and new objects take a long time before reaching the highest score.

From this observation and current informal experiments, we believe that, ideally, the filtering properties of the score accumulation function should match the context of the scene and the type of objects. We are currently investigating these filtering properties in more detail and, and we hope to discover to what extent automated tuning based on scene context can provide improvements in various selection scenarios.

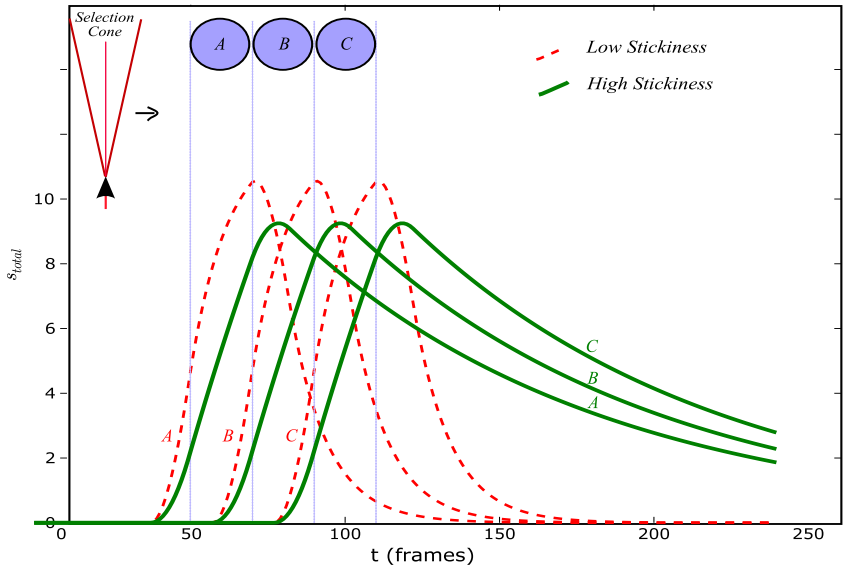


Figure 3.9: Scoring response of three static objects in line while the selection volume moves along them: The scoring response parameters influence the selection timing. The high stickiness setting spreads out the score over time, effectively providing more time for user interaction.

3.4.5 Scoring redistribution

In most VR application a hierarchy of objects is used to create complex objects while maintaining flexibility in scene management. Often not all parts are useful for selection or manipulation, but only a specific object is. For selection, the nested bounding boxes of a tree of objects and sub- or child- objects might be used, where the parent object's bounding box contains all the bounding boxes of its children. In our original IntenSelect scoring metric we have not taken this object hierarchy into account, which can make the selection of sub-objects contained in other bounding boxes rather difficult. In addition, we want the selection of a parent object to be able to trigger the selection of a sub object. To facilitate this we provide scoring redirection. Here, the scoring value obtained by the parent object(s) in the tree can be redirected to some or all child objects. Especially in situations where large objects have only small selectable and manipulable sub objects, this redirection can be used to trigger the sub objects on selection of the parent object. A useful example of nested UI elements on which we applied redistribution, are our window and widget constructs, shown in Figure 3.10.

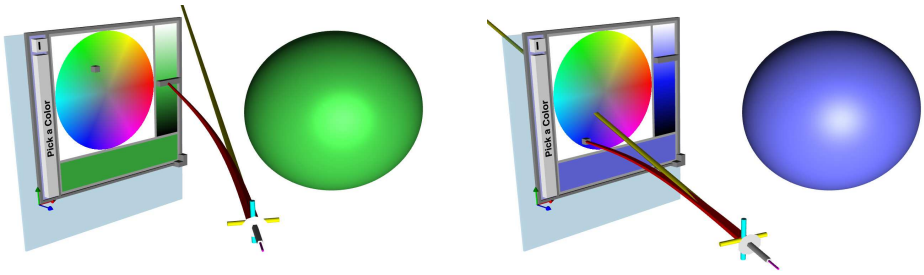


Figure 3.10: Color picking on the PlexiPad: supported selection of the small controllers, while manipulation is dynamically constrained.

3.5 Results: Integrating Interaction and Interface

The individual windows, dialogs and control elements are constructed from regular VR objects that work directly with our supported interaction mechanism. To improve the selection and manipulation behavior of our interface elements, we used the interaction extensions described above. In this section we describe the most notable integration results we obtained in several small test applications. We conclude with a description how the bulk of the material integrated into our Cloud Explorer application.

3.5.1 VR System Characteristics

We will briefly overview the characteristics of our primary VR system used in this work. For hardware, we make use of a Infitec-based passive stereo Virtual Workbench setup with a 180×110 cm screen size and a resolution of 1400×860 pixels. For tracking we use a Polhemus FASTRAK electromagnetic tracker tracking 4 devices, each tracked interleaved at 30 Hz. Our system is powered by 2 dual Intel Xeon Linux-based rendering machines. On the software side of our implementation, we worked with OpenGL Performer and an in-house VR library. We make use of VRPN for monitoring our tracked devices, and we have implemented predictive filtering to help compensate for tracking latency.

3.5.2 Snapping and Constraints

In Figure 3.10, we illustrate the use of constrained manipulation in the case of the color picker dialog. As described in section 3.3.3, the dialog's two controller elements can be used to control the selected color value. The entire dialog and window can be repositioned and resized in 3D space, in this case on the PlexiPad, while the controllers' movements are actively constrained to their respective control regions. At the left of Figure 3.10, the controller manipulation is limited to a 1D movement

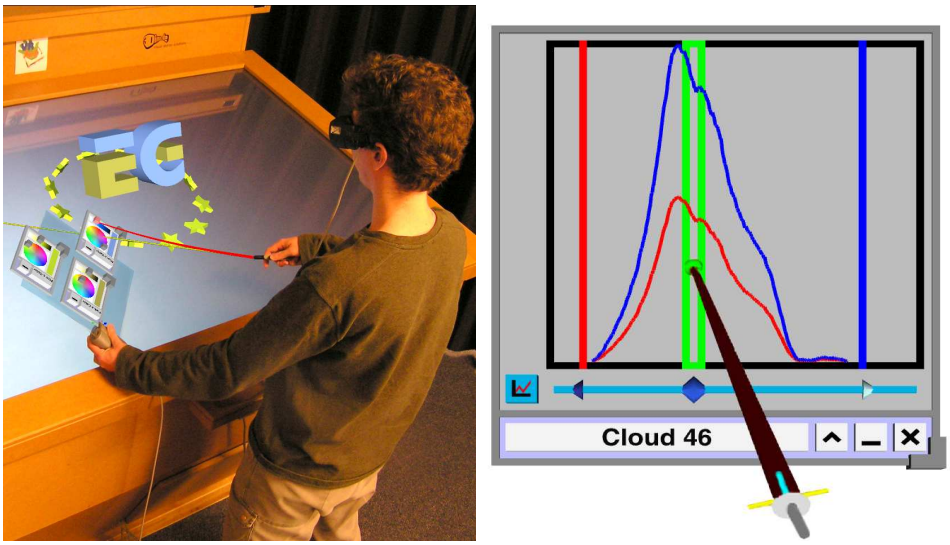


Figure 3.11: Left: Two-handed interface control of windows on the PlexiPad. Right: Ray-based snapping point on the Graph window provides precise a 2D interaction point.

over the lightness bar. At the right of this figure, the second controller is manipulated and restricted to the circular hue/saturation 2D region. In both situations, if the controller movement is restricted by constraint boundaries, the ray is deformed to maintain the flexible connection. Figure 3.11 (left) shows a user controlling these small dialogs on the PlexiPad. As the windows are fixed to a particular plane in space, in this case the PlexiPad, we use similar constraints on the windows during manipulation. Once the widget is being manipulated, the flexible deformed ray will again maintain its connection, also if stylus and PlexiPad are moved.

As discussed, for some interaction we need precise interaction on information panels and windows. To illustrate this we use ray-based snapping mode in the graph panel to extend the widget based control, see Figure 3.11 (right). Users can directly select a 2D position in the graph layout, which is used, in this case, for updating the current time slider.

3.5.3 Selection and Readability

Due to tracker miscalibration, the passive haptic feedback provided by the PlexiPad and the Workbench display did not always work as expected. Although the supported direct manipulation reduced the severity of this problem, selection of small cluttered objects remained difficult due to tracker error. This was doubly the case if objects were placed on the PlexiPad, thus requiring two tracked devices to work

together. To help solve this problem, we can use a higher stickiness setting for small individual widgets, such as the buttons in the numeric input dialog. Careful parameter selection provides a subtle smoothing of the jitter for in these scenarios.

IntenSelect permits scaling of widgets down to sizes of only a few screen pixels. On the Workbench we have scaled down the elements as far as text legibility allowed. In this configuration, the text height on the buttons was limited to 10 pixels, which corresponds to about 12mm on our screen, which is well readable. We must note that these dimensions are dependent on the screen resolution and viewing distance, but foremost the font type and anti-aliasing quality. Using these widget sizes, we experience no difficulty selecting the interface elements. Manipulation was also intuitive and easily accomplished, but accurate manipulation at a distance still relies, to some degree, on the user's manual dexterity.

3.5.4 Integration with Cloud Explorer

The use of our new hybrid interface and interaction technique have made a significant difference in the UI for Cloud Explorer. Figure 3.12 shows a comparative illustration between the old UI and the new UI. While most of the basic elements from the old UI remain, they are now much smaller and easier to use. The size is critical because Cloud Explorer is still in its infancy, and, yet, the interface is already quite cluttered. With the new UI, irrelevant portions of the interface may be hidden, and various portions can be moved to accommodate the user's preferences (e.g. left-handed users). With the improved interaction, the buttons, sliders, and clouds are all easier to use. We were surprised at how easily the sliders on the graph windows, being around 15×15 pixels, could be selected and manipulated at distances of over one meter. Direct interaction with UI elements is simple and intuitive in spite of a persistent tracker miscalibration. The addition of the numeric input dialog also addresses a common user complaint for selecting clouds that have died out and are no longer visible. Furthermore, the transition between interacting with the UI elements and the clouds themselves is seamless.

3.6 Conclusions and Future Work

The use of hybrid interfaces is important in our application area: scientific visualization. We have demonstrated two techniques we developed and integrated to address some of the limitations of hybrid interfaces.

We implemented a new hybrid interface, which offers users a less cluttered and more flexible UI. It integrates command functionality and enriching information in an intuitive manner through the use of the familiar windows paradigm. We make use of constructs such as tooltips and dialogs to maintain a lightweight UI without alienating the novice user.

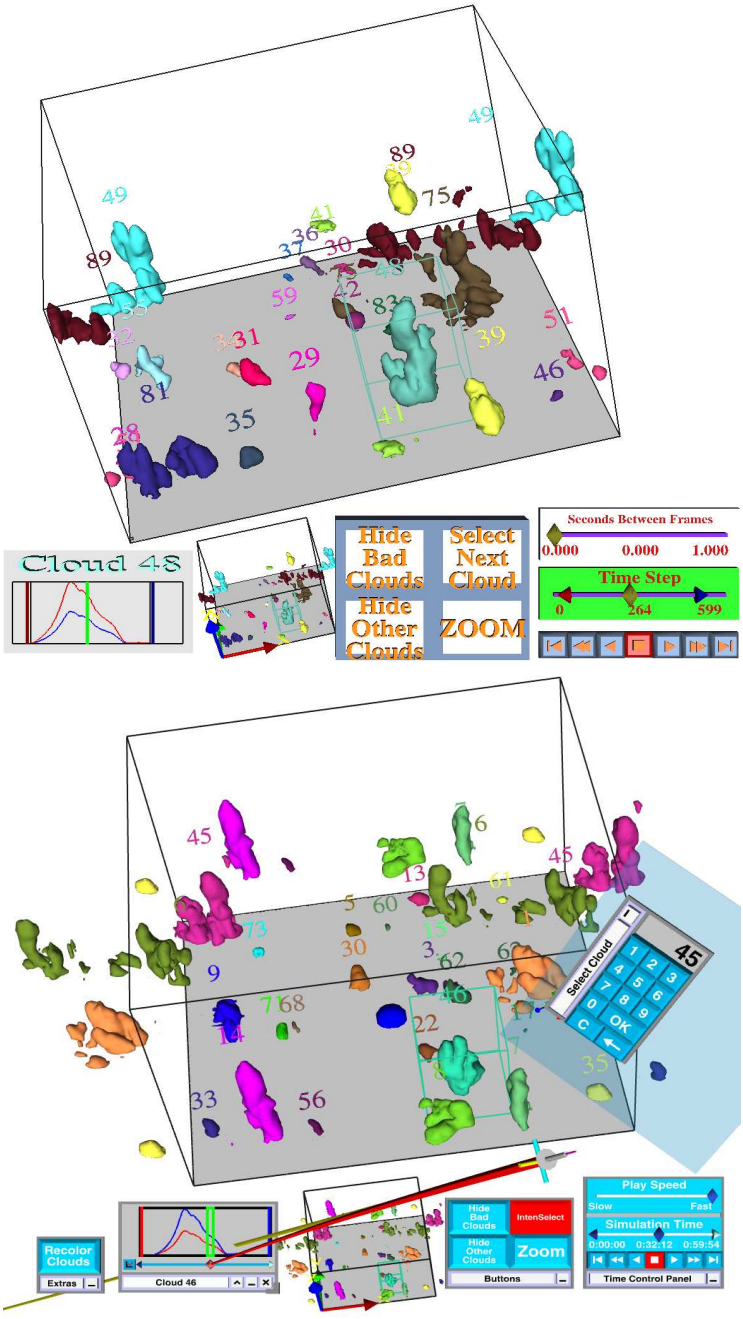


Figure 3.12: A comparative illustration between the old Cloud Explorer interface (top) and the new one (down). Most of the basic UI elements are now smaller but easier to use.

The new extensions to our IntenSelect method provide the user with more intuitive and easy to use tools to interact with the VE. The use of generic scoring metrics and filtering provided a flexible framework for implementing special interaction behavior. Direct selection and improved object scoring makes it easier for the user to select objects of interest in various scenarios. New object snapping and object manipulation techniques allow the user to effect meaningful changes in the virtual environment with less effort.

We have used our Cloud Explorer application as an example of the kind of UI that this integration made possible. The environment is filled with an array of 3D objects and 2D UI widgets, and the transition between interacting with each is seamless. The UI affords more room for the 3D clouds, while also giving the user more flexibility to arrange it to his or her own taste. Less relevant elements can be positioned at the edges of the Workbench without presenting any difficulties to the user if he wishes to read or interact with them.

As the need for more abstract and complex interaction and information panels grows in our exploration applications, we can extend our interface elements to control its specific interaction behavior in more detail. We believe that the integrated solution of flexible interfaces and interaction support allows us to stretch the complexity limit of interface scenarios in VR, without usability issues exploding.

We would like to continue to extend and enhance our hybrid interface in two ways. First, we would like to develop new and easier to use widgets, and offer more intelligent interface management and placement. The latter is useful due to the erratic behavior of overlapping 2D elements. Secondly, we would like to use the interface in various scenarios such as multiple-linked views. Here, the need for a convenient method for managing global state variables through bi-directional interface elements will be necessary.

We continue to extend and develop the (mathematical) foundations of IntenSelect towards improved scoring behavior in various situations. As described earlier, we hope to discover to what extent automated tuning based on scene context can provide improvements in various selection scenarios. Furthermore, we plan to extend our scoring and snapping mechanisms to facilitate cooperative interaction for multiple users and interaction devices.

To strengthen our statements on usability and limitations of our techniques, we plan to fine-tune parameters and perform user tests on a wide variety of VR systems, of which a Personal Desktop VR system and a large CAVE-like system are candidates.

4

Consistent Viewing and Interaction for Multiple Users in Projection-Based VR Systems

This chapter contains a slightly modified version of an article published earlier. The original peer-reviewed article appeared in Computer Graphics Forum [de Haan 07c] and was presented at the EuroGraphics Conference in Prague, Czech-Republic, 2007.

Overview

In projection-based Virtual Reality (VR) systems, typically only one headtracked user views stereo images rendered from the correct view position. For other users, who are presented a distorted image, moving with the first user's head motion, it is difficult to correctly view and interact with 3D objects in the virtual environment. In close-range VR systems, such as the Virtual Workbench, distortion effects are especially large because objects are within close range and users are relatively far apart. On these systems, multi-user collaboration proves to be difficult. In this chapter, we analyze the problem and describe a novel, easy to implement method to prevent and reduce image distortion and its negative effects on close-range interaction task performance. First, our method combines a shared camera model and view distortion compensation. It minimizes the overall distortion for each user, while important user-personal objects such as interaction cursors, rays and controls remain

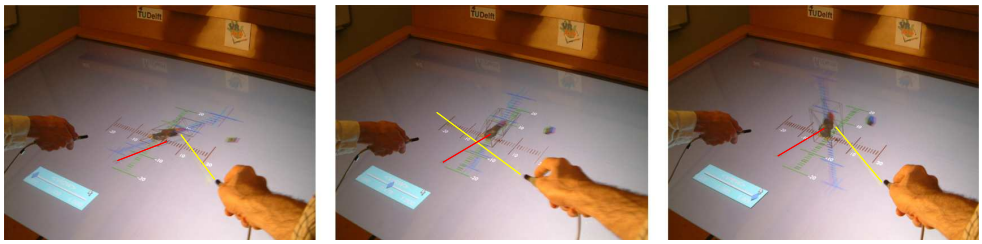


Figure 4.1: Demonstration of our method in a two-user scenario at the Virtual Workbench. The scene is rendered from the viewpoint of the left user (left image), the right user (right image) or a dynamic, average viewpoint (middle image). Although objects may not appear in the correct perspective for either user in the averaged viewpoint, interaction with the distorted environment is still possible, as each user has personal, corrected interaction tools.

distortion-free. Second, our method retains co-location for interaction techniques to make interaction more consistent. We performed a user experiment on our Virtual Workbench to analyze user performance under distorted view conditions with and without the use of our method. Our findings demonstrate the negative impact of view distortion on task performance and the positive effect our method introduces. This indicates that our method can enhance the multi-user collaboration experience on close-range, projection-based VR systems.

4.1 Introduction

Large screen projection displays, such as the Virtual Workbench, invite direct collaboration between a small group of people. The shared, interactive viewing of information on a single screen with multiple users facilitates natural communication on the virtual environment. The various 3D depth cues used in rendering a correct 3D image of the information are only optimal when observed from a single viewpoint. Users that are not close to this optimal viewpoint perceive a distorted graphical representation, which has a negative influence on system usability and collaboration.

Two general approaches have been proposed to enhance collaborative use of projection-based Virtual Reality (VR). First is the use of additional tracking and display hardware to generate correct (stereo) images for each user individually [Bolas 04]. This approach involves costly extensions to standard projection-based systems and image quality often suffers from limitations in image separation techniques. The second approach applies extra tracking hardware and special rendering techniques to cope with image distortion [Simon 07]. Image rendering and depth cues are adapted to reduce the overall perceived distortion.

Although this solution does not provide a completely correct image for all users, we still expect it to be very usable in real collaborative scenarios. Furthermore, as up-scaling tracking hardware for more users is easier and more cost-effective than

additional display hardware, it would provide an attractive option for extending new and existing projection-based VR systems. For these reasons, we chose to analyze, evaluate and extend this solution for use on close-range VR systems, and, more specifically, on our Virtual Workbench system, where users stand relatively further apart, have a very different, off-axis viewing perspective and use direct and co-located interaction. Such a collaborative two-user scenario at the Virtual Workbench is illustrated in Figure 4.1.

The contribution of this chapter consists of three main parts. First, we analyze of the distortion effects and their influence on 3D interaction and co-location. Second, we present a method to minimize view-distortions of individual users while maintaining co-location for effective interaction. Finally, we present a user study performed with our Virtual Workbench, in which test subjects performed 3D interaction tasks using both direct and remote techniques. The experimental results quantify the negative impact of view distortion and the corrective effects of our method. This chapter is constructed as follows: We first describe related work in section 4.2. Then, in section 4.3, we describe the causes of distortion effects in projection-based VR and our approach for a solution. This is followed by a technical description in section 4.4. Our method is evaluated during a series of user experiments, of which relevant results and an analysis are presented in section 4.5. Finally, we draw conclusions and discuss possible future extensions in section 4.6.

4.2 Related Work

A general challenge for multi-user 3D stereo displays is to provide a separate, correct image for each eye of each user [Bolas 04]. Besides tracking each user's head, the key challenge here is the separation of all the images. For projection-based VR system, solutions include optical filtering (e.g. polarization filters), time multiplexing (e.g. shutter glasses), multiple screens, or a combination of these. The techniques can be used to extend existing, single-user systems, or to construct new multi-user systems from scratch.

The use of time multiplexing is demonstrated in the *Two-User Workbench* system [Agrawala 97] and in a multi-screen immersive environment [Blom 02]. Limitations included decreased image brightness, crosstalk and image flicker. A hybrid solution is demonstrated in [Fröhlich 05], where polarization filtering and time multiplexing was carefully combined to achieve better image quality for more users. Separate screens, masks and mirrors can be used to create a shared physical space for all users, for examples the *PIT* [Arthur 98] and the *Virtual Showcase* [Bimber 01]. Technical limitations in display hardware and physical installation issues limit the applicability and scalability of extending existing VR systems for collaborative use for multiple users.

Simon et al. describe *multi-viewpoint merging*, an alternative approach to cope with multi-user interaction on a single display [Simon 05][Simon 07]. Instead of gen-

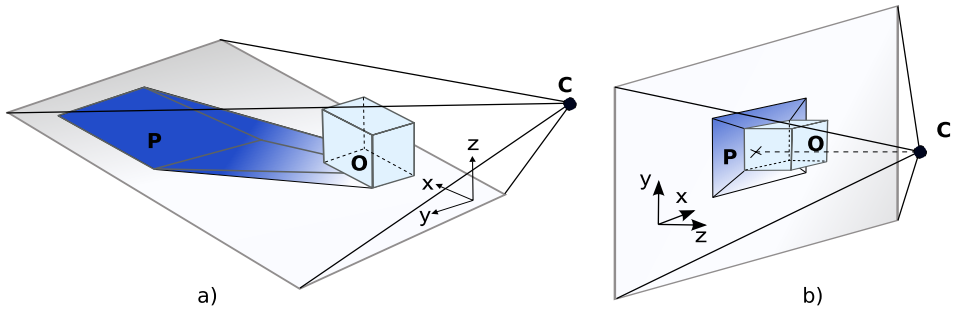


Figure 4.2: Difference in projection transformations: The Virtual Workbench (a) has an asymmetric off-axis frustum, while a Virtual Wall (b) has a symmetric on-axis frustum.

erating a completely correct 3D image of the entire virtual environment for each user, the scene is rendered from a static, central viewpoint. Only some elements in the scene, like interaction tools, are displayed correctly for its corresponding, head-tracked user by correcting the visible projection of these objects. They report that, although many objects are visually distorted, this solution still remains usable for collaborative work in VR systems. At the cost of image distortion and object deformation, existing systems can be extended for multiple users by tracking the extra users. These interesting properties were our main motivation to extend this work at three points.

First, as the Virtual Workbench differs greatly from the panoramic screen used in Simon's study, we suffer from larger distortion effects. In close-range, tabletop VR systems users stand relatively further apart and have a very different, off-axis viewing perspective (see Figure 4.2), which is sensitive to head motion. Also, virtual objects are mostly within hand reach, and direct, co-located manipulation techniques are preferred. Therefore, we perform a problem analysis to investigate the source and effects of this distortion on interaction in more detail, as [Wartell 99] did for eye-separation. Second, roles among users [Heldal 05a] as well as proper division of tasks are important factors that influence the success of collaboration and task performance. We strive for a flexible solution to match the quality of viewing and interaction with the task and roles in mind for each user. For this we investigate the advantages and disadvantages of different solutions and camera options. Third, the question arises how much distortion can be tolerated in various collaborative VR scenarios. Only a few user studies exist concerning multi-user setup and distributed multi-user VR systems, for example [Heldal 05b]. With the benefits and limitations of different quantitative and qualitative evaluation in mind, we conducted a user study on our Virtual Workbench.

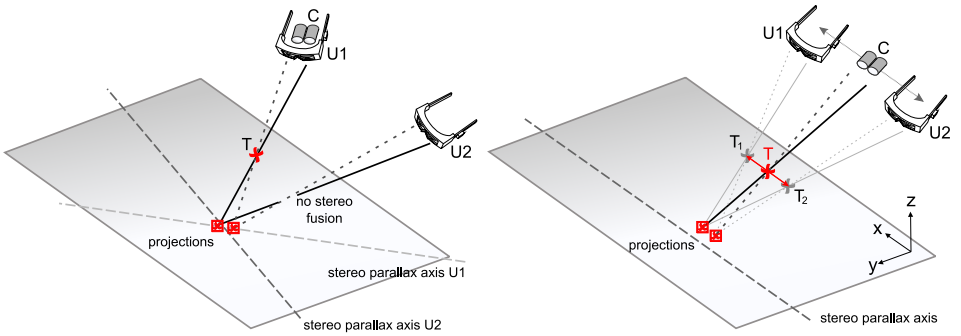


Figure 4.3: Two-user viewing problems. Images are rendered for User1 ($U1$). Depending on the head pose of both users, the stereo images may be difficult or impossible to fuse for User2 ($U2$) (left image). This is caused by $U2$ having a different stereo-parallax axis. To avoid parallax differences between users, the cameras can be constrained to be parallel to the x-axis. $U2$ will now perceive stereo, although point T is perceived at location T_2 (right image)

4.3 Analysis and Approach

We performed a usability analysis to investigate current problems and possible solutions [Molenaar 07]. Figure 4.3 illustrates the basic problems of a collaborative two-user scenario on a regular workbench system, while the usability claims and the most important advantages and disadvantages are summarized in a table.

4.3.1 Problem Description

In the classic, single-user scenario, only one user (User1) experiences both the stereo and motion parallax depth cues correctly. For another user (User2) viewing the same screen, interaction can become much more difficult or even impossible.

First, the stereo fusion may be lost. When the User1 tilts his head, the stereo parallax remains correct only for him, but becomes very unpleasant for User2. Depending on the head tilting of User1 it may even become impossible for User2 to perceive depth, as there will be no intersection between the lines of sight to the two projection points that are to be fused.

Second, if User2 does perceive stereo images, he still experiences both point of view distortions and motion distortions. Point of view distortions are the result of seeing the projections from a wrong point of view (see Figure 4.3); an object may appear sheared and at the wrong location, and interaction tools do not appear co-located either. Also motion distortions influence interaction and are the result of the absence or presence of parallax effects. Head movements of tracked users result in image changes, while non-tracked users do not experience the motion parallax effect when they should.

	General claims if User1 - headtracked; User2 not
+	Face to face and non-verbal communication possible
+	Shared use of system resources
+	Several users are able to add value to the cooperation
-	Hardware supports only one (active) user
-	Tracked glasses and interactions devices may need to be switched, this interrupts the session
-	Some movements of User1 cause eye strain for User2
-	Some movements of User1 cause in-fusible images for User2 (see Figure 4.3)
	Point of View Distortion for User2
-	Objects will appear sheared
-	Objects will appear at a wrong location
-	Users need to compensate for distorted view during interaction
	Motion Distortion for User2
-	User1 head movement will cause the objects to move, as seen from User2's point of view
-	Head movement User2 will not update the world
-	Interaction is disturbed by User1's head movements

In our research we explored several solutions with alternate forms of display and headtracking as well as direct, close-range manipulation. In this chapter, we only concentrate on one approach which focuses on the interaction problems that occur when looking at a single scene rendered from an incorrect viewpoint.

In our effort to find a suitable solution for multi-user use of a single rendering, the overall quality of usage and the performance of additional tracked users is taken into account. One goal is to provide equal opportunities for all users, causing two users to be able to collaborate equally and also have better sense of what the other user is seeing [Heldal 05a]. To support multiple users on one system we use alternative camera models, which are presented in this section. Besides choosing a camera model we can also solve some interaction problems by calculating proper locations of interaction elements of other tracked users.

4.3.2 Alternative Camera Models

Some viewing problems can be eliminated by altering the effects of headtracking on the rendering viewpoint. In some scenarios, eliminating headtracking completely can make the system more accessible for multiple users. A main advantages is that adding additional viewers will not be limited by hardware limitations. Images remain stable and fixed, so no user will experience the motion swimming distortion and there will be no in-fusible images caused by the head movements of a single user. Of course, there is only one specific position to have the correct perspective,

and the elimination of motion parallax takes away important depth information. In general, this solution is best for larger audience presentations (e.g. a panoramic screen [Simon 05]), and not for close-range, interactive work. For the Virtual Workbench, the elimination of all motion parallax takes away much of the depth perception. Also head tilting is a greater issue here.

Instead of totally eliminating all headtracking we can also partly constrain the headtracking. By only tracking head position and not the rotation, the headtilting problem can be reduced. Since the stereo images will be rendered with horizontal parallax only (some systems only allow horizontal parallax due to the passive filter technique used). Additional viewers will have less fusion problems when the headtracked user rotates his head. But the headtracked user is now unable to tilt head and keep images fusible for himself. Users will quickly learn however how to hold their head. As a result the images will remain more stable.

The distortion can be split equally between users, by placing the viewpoint at an average position between the multiple headtracked positions, see Figure 4.3. By using this *dynamic average viewpoint*, two users will experience half of the distortions, but also half of the motion parallax, which can still serve as an important depth cue. Both users will experience some motion swimming effect. Head rotations will not be taking into account, but there will be reduced possibility of eye strain and in-fusible images.

4.3.3 Viewpoint Compensation

When choosing an altered camera model, users will observe distortions in both perspective and motion parallax of the projections. The need for users to manually compensate for this when using interaction devices can be counteracted. By using the headtracking of a user, interaction can be based upon his point of view. This is accomplished by pre-distorting or *warping* the geometry of some (interaction) elements of the scene in such a way that they seem correct from his specific viewpoint. While these elements can be shown correct only for that user, the rest of the scene can be updated according to the active camera model, such as the dynamic average viewpoint. We call this *viewpoint compensation*. This is similar to the *multi-viewpoint merging* solution [Simon 05], where selection rays are displayed correctly for a specific user while the overall scene is projected for a static viewpoint. In our dynamic average viewpoint approach however, we also need to apply continuous compensation to counteract additional motion swimming distortions. In this way, perception and action can be made consistent for every tracked user even though the overall scene may appear distorted by perspective and motion parallax effects, see Figure 4.4(left). This warping process and its implementation is described in the following section.

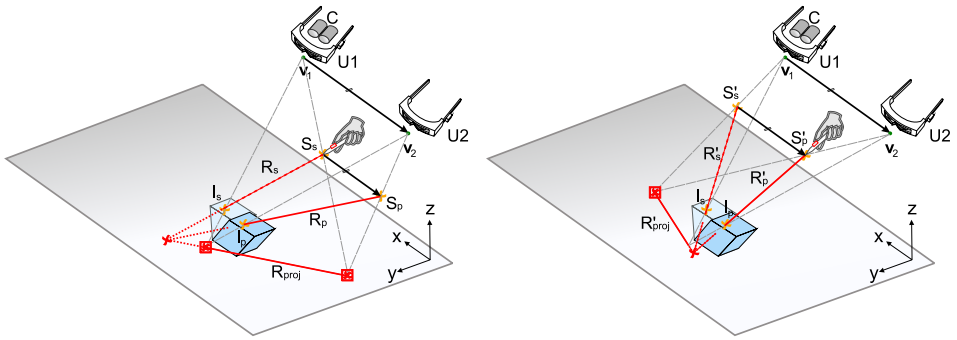


Figure 4.4: View distortion during ray-casting (left) and with viewpoint compensation (right). The camera is at $U1$'s viewpoint, subscripts $S, P, Proj$ stand for in scene, perceived and projected. User $U2$ holds a stylus and tries to shoot a ray on the cube. In the left image, $U2$ experiences inconsistency: based on R_{proj}, he perceives R_P as the ray coming from S_P. In the right image, we correct for viewing distortion by pre-transforming the ray to R'_S. From the new R'_{proj}, $U2$ now correctly perceives the ray R'_P as shooting out of his stylus S'_P, and can consistently point at the perceived cube.

4.4 Method

In this description of our method and its implementation in our VR system, we distinguish between the consistent viewing of the scene and consistent interaction with objects in the scene. We define consistency as the agreement between the user's actions and his perceptions. To clarify this in the context of distorted spaces, we distinguish three different coordinate spaces. Figure 4.5a illustrates the three coordinate systems used. In the *tracker space*, the tracker hardware registers the actions performed by the users. Correct alignment and calibration allow a direct translation of these measurements to the *scene space*. This scene space is rendered and projected onto the screen. Based on these projections, each user makes a mental 3D reconstruction of the scene, which we call the *perceived space*.

In a normal situation, the rendering viewpoint directly corresponds with the user's head position. Therefore, the tracking space and the perceived space directly agree, providing a co-located virtual environment. When these projections are observed from a different position, the mental 3D reconstruction of the scene or perceived space does not agree anymore with the tracker space. Since we know how this tracked user perceives the scene, we can apply appropriate calculations to make actions and perception consistent again. These calculations consist of corrections for viewing and corrections for interaction, see Figure 4.5b.

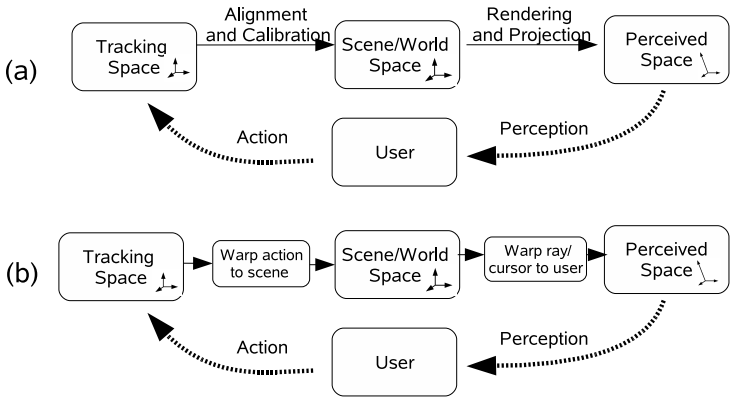


Figure 4.5: Relations between coordinate spaces: (a) the user observes the scene in perceived space, while performing actions in tracking space. (b) When perceived space and tracking space are not in agreement, corrections on both visible geometry and the performed actions are needed.

4.4.1 Consistent Viewing

Any element of the scene that is directly related to a certain user can be transformed to match the perspective of the user. We will describe necessary algebraic calculations for this *warping* process.

We first transform the interaction rays, cursors and other private elements to the perception space of the user, see Figure 4.4. In this way, we can display these elements co-located with users interaction devices. For example, we can make the interaction ray seem to shoot out of the stylus, or correctly display widgets on our *Plexipad*, a handheld transparent prop.

When using this method, all interacting users need to be headtracked. In our Virtual Workbench system, we use six 6-DOF sensors, to fully support two users. Users are headtracked and they have a stylus and a Plexipad. The headtracker is used to register the users viewpoint positions $\mathbf{v}1$ and $\mathbf{v}2$, see Figure 4.4. The current positions of $\mathbf{v}1$ and $\mathbf{v}2$ are used to calculate the amount of perspective distortion for each user. When for example the images are rendered from a camera at viewpoint $\mathbf{v}1$, $U1$ correctly perceives the scene while $U2$ observes a sheared scene.

We use a *warping matrix* to properly transform those elements in the scene that are associated with a specific user. This warping matrix allows us to transform points and geometry between the perceived space and the scene space. Figure 4.6 illustrates this situation. User $U1$ has a correct view and observes the Z axis as being perpendicular to the ground plane, pointing upwards. His perception of the scene space can be described by an orthonormal coordinate system matrix, which is in this case equal to the identity matrix, as all the axis base vectors are orthogonal and have a unit length. The view of the second user $U2$ however, suffers from a shearing dis-

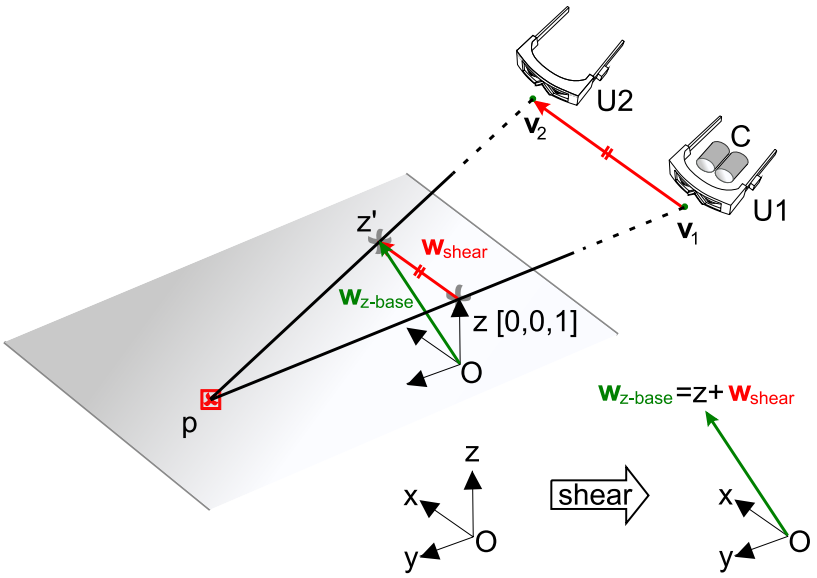


Figure 4.6: Construction of the sheared coordinate system: The axis vectors X and Y are not distorted. The Z axis suffers from shear distortion. Point Z is perceived at Z'.

tortion. He experiences this axis as pointing into a sheared direction. The X and Y base vectors remain unaffected, but the Z base vector is affected.

The vector $\mathbf{w}_{shear}$ defines the magnitude and direction of the shearing and can be calculated from any point above the ground plane. Here, we use the base vector $\mathbf{z}[0, 0, 1]$ for convenience. As shown in Figure 4.6, its projection from viewpoint $\mathbf{v}1$ is the point $\mathbf{p}$. We calculate $\mathbf{w}_{shear}$, which is parallel to the vector $\mathbf{v}1\mathbf{v}2$, see equation 4.1. Then, we calculate position of the point $\mathbf{z}'$ in the perceived space of the user $U2$, see equation 4.2.

$$\mathbf{w}_{shear} = \frac{|\mathbf{z} - \mathbf{p}|}{|\mathbf{v}1 - \mathbf{p}|}(\mathbf{v}1 - \mathbf{v}2) \tag{4.1}$$

$$\mathbf{z}' = \mathbf{z} + \mathbf{w}_{shear} \tag{4.2}$$

We use the point $\mathbf{z}'$ to calculate the vector $\mathbf{w}_{z-base}$, which is the z base vector of the coordinate system of the second user, see equation 4.3.

$$\mathbf{w}_{z-base} = \mathbf{z} + \mathbf{w}_{shear} \tag{4.3}$$

The amount of the distortion for the whole scene can be encapsulated in the shearing matrix $\mathbf{M}_{shear}$, as shown in equation 4.4. Basically, we insert the vector $\mathbf{w}_{z-base}$ at

the place of the normal z-base vector.

$$\mathbf{M}_{shear} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \mathbf{w}_{z-base} & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

It is clear that in this approach, the shearing distortion affects the rendering of the individual vertices of geometry in the scene. The amount of shearing for each vertex is a function of the camera position $\mathbf{v}1$, the observer position $\mathbf{v}2$ and the z coordinates of the vertex. In general: the higher a point, the larger the shearing distortion, the larger the distance between $\mathbf{v}1$ and $\mathbf{v}2$, the larger the shearing distortion. The correction matrix $\mathbf{M}_{warp}$ for counteracting the distortion, is obtained by inverting $\mathbf{M}_{shear}$, see equation 4.5.

$$\mathbf{M}_{warp} = \mathbf{M}_{shear}^{-1} \tag{4.5}$$

This matrix can be used to *pre-distort* or *pre-warp* geometry in scene space. We implemented the warping function as a dynamically updating scene graph node, which can be used to easily correct distortion effects on various parts of the scene.

4.4.2 Consistent Interaction with Scene Objects

Having only a proper visual perception is not yet sufficient for consistent interaction from a different viewpoint. Although the perceived space is made to match the scene space, it does not directly match the tracking space that registers all interactions. Theoretically, these spaces can also be made to agree by applying a correction, similar to the previous section. However, implementation details of interaction techniques and scene graph hierarchies prevent this, and would result in unexpected deformations of objects. Interaction techniques typically use transformation matrices and quaternion algebra, but assume orthonormality of the input data. As soon as we would apply distortion corrections on tracker data, orthonormality would be lost. As a result, shearing aspects would interfere with normal calculations and would introduce unexpected deformation in objects in scene space. To avoid these distortions we describe the adjustments needed to use interaction techniques under distorted viewing conditions.

Consistent Selection

For object selection, we use both direct and remote (ray-based) techniques. The *direct selection* technique uses a single 3D point provided by the stylus. On account of possible viewing distortions, we need to multiply that point in percieved space S'_p by $\mathbf{M}_{shear}$ matrix to obtain the corresponding point in scene space S'_s. The *ray-based selection* technique uses both a point and a direction vector provided by the stylus,

both of which have to be transformed from tracking space into scene space. The (unit) direction vector can be extracted from the orientation quaternion of the stylus $\mathbf{q}_{stylus}$. To avoid distortion effects that could occur by warping the quaternion directly, we divide the calculation of the ray in the transformation of two points. For this, we calculate the position of an arbitrary second point along the stylus ray. The two points, the stylus tip and the second point, will be converted into scene space. Based on these two points, the correct selection ray can be reconstructed in scene space.

Consistent Manipulation

For object manipulation, we also use both direct and remote (ray-based) techniques. Once the object has been selected, both techniques use the same algorithm for manipulations. The important adjustment here is the separate treatment of translation and rotation transformations. The reason for this is that the translation matrices remain pure translation matrices despite of shear distortion, for rotations this is not the case. A more detailed technical discussion on maintaining consistency during manipulation, and especially rotations, is given in [Koutek 07].

The pose of an object consists of a position (x, y, z) and orientation (quaternion $\mathbf{q}_{object}$). After object selection, we calculate the so-called *interaction point* I_s, see Figure 4.4. We store the distance vector R'_p between I_p and stylus position S'_p and the vector $\mathbf{v}_{OIs}$ between the objects origin and I_s. For direct manipulation, distance R'_p is zero, while for ray-based manipulation this corresponds with the length of the selection ray. During manipulation, we combine the stored $\mathbf{v}_{OIs}$ and R'_p with the new stylus position S'_p to translate the object to the new positions in the scene. To calculate the new orientation of the object during manipulation, we first store the quaternion difference $\mathbf{q}_{delta}$ between the initial object orientation and stylus orientation $\mathbf{q}_{stylus}$. Then, every manipulation step the current quaternion $\mathbf{q}_{stylus}$ is warped to match the intended rotation. In this way, we obtain $\mathbf{q}'_{stylus}$ in scene space. To calculate the new object rotation around the point I_s, $\mathbf{q}'_{stylus}$ is multiplied with $\mathbf{q}_{delta}$.

During manipulation two extra distortion effects become noticeable. First, in normal ray-based manipulation, the length R_s of the ray remains constant in scene space. However, distortion within the perceived space and head movements cause the amount of distortion in $\mathbf{M}_{shear}$, and thus the perceived length R_p of the ray, to constantly change. The correct length is calculated by taking the distance between the current stylus position S'_s and the unwarped interaction point I_s. Second, extra side effects occur when applying rotations on objects. Rotations affect the observed shape of the object, caused by the shearing that affects the object. Also, we can not directly apply rotational movements made with the interaction tools, since rotation angles change in sheared space. This is especially noticeable when the $\mathbf{M}_{shear}$ matrix is changed significantly and when upward and tilting movements are made at the same time. A detailed technical description of these extra distortion effects and its compensation are beyond the scope of this chapter and are given in [Koutek 07].

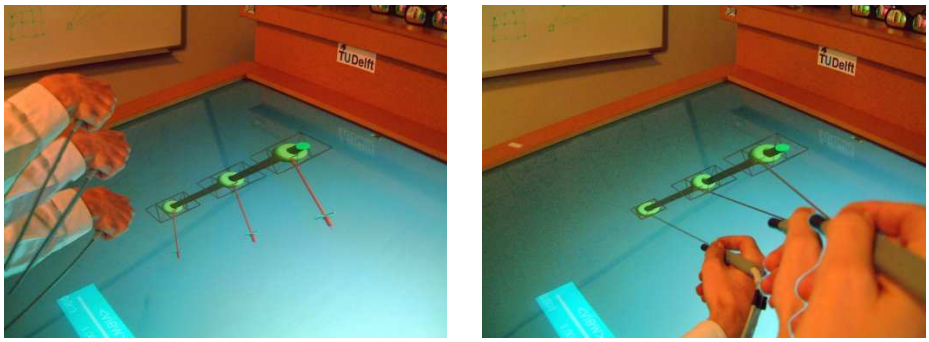


Figure 4.7: Sequence of ray-casting manipulation task. The scene is rendered from User1's viewpoint and photographed from User2's viewpoint. In the left image, User1 sees the ray shoot from his stylus (although User2 does not) and moves a ring upwards along a vertical cylinder. In the right image, User2 performs exactly the same task from a different viewpoint. By using warping, the ray shoots from his stylus and he can move the ring diagonally along the distorted cylinder.

4.5 Evaluation

A series of user experiments was performed for a qualitative and quantitative evaluation of our method. These experiments were performed on our Virtual Workbench system, a large tabletop stereo display combined with 3D tracking hardware. The table surface is slightly tilted at 10 degrees, measures 180 by 110 cm and provides room for multiple standing users. Stereo is provided by two projectors equipped with *Infitec* passive stereo filtering. We used two 6-DOF electromagnetic tracking sensors for each active user, one for headtracking and one for stylus tracking. The software for performing the experiments and evaluation was implemented in Python on top of our *iVR* software framework, see also Chapter 5.

4.5.1 Experiments

The experiments of interest here were part of a larger set of two-user experiments, consisting of evaluation of various display configurations and the evaluation of interaction performance. Each experiment took about an hour for each user to complete. The first part investigates the activities of pairs of users working side by side at the Virtual Workbench in different roles. The second part analyzes the impact of distortion on the task performance and accuracy of co-located selection and manipulation interaction techniques for a single user.

The first part consisted of informal discussions on various two-user display, tracking and interaction scenarios. Participants were asked to interact with the VR scene, to think aloud and to give their opinions about the distortions and proposed solutions. The scene consisted of randomly placed colored cubes. Users explored

interaction possibilities and communicated through questions like "What color is this cube?" and "Can you place that cube over here?". In each scenario, the quality of interaction and communication was discussed and the different techniques were explained. After these explorations, users were asked to fill out questionnaires consisting of Likert scale questions and short answer questions.

The quantitative part of the experiment was performed by each participant individually. Here, the direct and remote selection and manipulation skills of the test subjects were measured under various conditions. A series of four tasks was performed: Object selection with the stylus cursor (*SC*), object selection with the stylus ray (*SR*), objects manipulation with the cursor (*MC*) and object manipulation with the ray (*MR*). Target objects were placed pseudo randomly and not yet visible in the VR scene, just above the tabletop of the Virtual Workbench. For the selection task, 12 small spheres were placed in a grid, while for the manipulation task six poles with a ring around them were used. Two poles were placed axis-aligned, while the remaining poles were placed diagonally over two or three axes. The selection task was performed as follows: First, the user placed the stylus inside a 'stopwatch sphere'. When the user moved the stylus out of the sphere, one of the twelve target spheres appeared and the stopwatch started running. The stopwatch stopped when the target was selected and the stylus button pressed. Then, the user returned the stylus to the stopwatch sphere to continue with the next target. The manipulation task was performed in a similar fashion. Here, the user had to pick the ring on the start of the pole first, then hold the button and move the ring along the pole to its ending, see Figure 4.7

4.5.2 Conditions

Twelve individuals participated in the experiments. For many subjects, this was their first VR experience. For the individual quantitative measurements, each of the four tasks was performed with eight repetitions under eight different conditions. The eight test conditions were constructed from variation in viewpoint configuration and the use of warping compensation: normal headtracked situation (*ht*), non-headtracked without viewpoint offset (*not ht*), non-headtracked with x cm offset from the correct viewpoint (*25*,*50*,*75*), and these same offsets again while viewpoint compensation for interaction was enabled (*25+w*, *50+w*, *75+w*). The offset distances roughly match the distances in the positioning behavior of the secondary user with respect to the primary user. These situations are: looking *over the shoulder (25cm)*, standing directly *shoulder-to-shoulder (50cm)*, and standing comfortably *side-by-side (75 cm)*. The targets for selection and manipulation appeared in random order in the scene and the sequence of sub-conditions was also chosen randomly each time. For each of the four tasks, the participant had a training period of two minutes under various, random viewing and interaction conditions. This training period was introduced to reduce the strong learning effect, which influences the performance of the individual tests.

4.5.3 Results

In this section we summarize the most important results on viewpoint compensation. A complete transcript of all experiments can be found in [Molenaar 07]. The main quantitative results are summarized in Table 4.1 and Figure 4.8. Important results of the qualitative evaluation are given here.

Users' reactions were very positive during the qualitative evaluation. We see from questionnaire results that users acknowledged that the system should allow multiple users to be active, and that proper alignment of their private tools created a more workable and pleasant situation. The possibilities for head rotations for one user was indeed a source of discomfort for the other user. They did not find the introduced constraining of stereo-parallax very discomforting. Some users needed to get used to not tilting their head too much. Users strongly agreed that the possibilities to communicate about the objects displayed and work together were enhanced, and that viewing compensation is essential when using interaction devices under distorted viewing conditions.

Users responded extensively in the short answers and confirmed that we made good improvements for collaborative situations. They remarked that the uncorrected situation was sometimes frustrating, and that viewing compensation made tool behavior predictable again, making tasks easier. One user stated the results clearly: "Seeing your cursor in the right perspective gives a feeling of control, that the space around it is distorted is not such a problem". Another user remarked that working with a wrong viewpoint is not that bad if it remains static, allowing users to adapt to the situation. A correct cursor made it a lot easier to adapt to a distorted viewpoint, and could also enhance the feeling of immersion. A steady image is more important than a correct image, and correct feedback is very important. Another user remarked that, without viewpoint compensation, it is nearly impossible to correctly collaborate, because one of the users will have difficulty operating interaction tools properly. Users did report some other, unexpected issues such as image blur at screen edges, light reflections and focus problems. For some, shorter users, targets would sometimes be outside the working area, preventing them from accomplishing a task normally. This shows that when large distortions occur, tasks can become difficult for other reasons.

From the quantitative results given in Table 4.1 and Figure 4.8, we observe the expected, negative effects of increasing distortion on both mean task completion times and errors when no viewpoint compensation is applied. Also, the standard deviation increases, indicating an increase in task difficulty. When compared to the performance of the classic, headtracked solution, only an offset of 25cm (looking over the shoulder) appears to be a comparable, workable solution. When standing shoulder-to-shoulder (50cm) or at comfortable distance (75cm), the mean selection time and mean error quickly raise up to and over 50%.

When viewpoint compensation is used, the negative influence of increasing distortion on task performance is not strongly visible. When compared to the head-

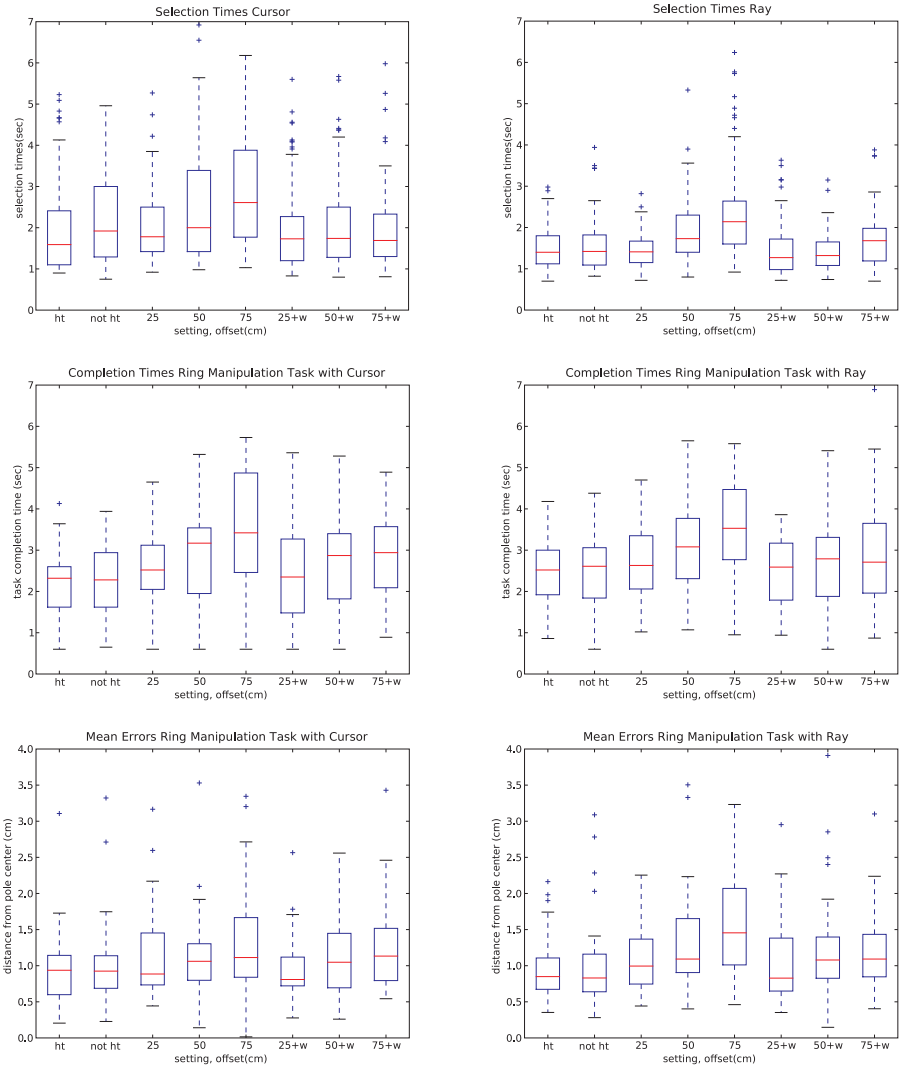


Figure 4.8: Experiment results. Left column: cursor (a) selection time (SC), (b) manipulation time (MC), (c) manipulation mean error (MCE). Right column: ray-based (d) selection time (SR), (e) manipulation time (MR), (f) manipulation mean error (MRE). Each sub-figure has eight box plots for viewing conditions, ht: headtracked, 25, 50, 75 cm offset from correct head position, and 25+w, 50+w, 75+w cm offset with viewpoint compensation.

Table 4.1: Experiment results. Selection time mean (M), standard deviation (SD), and percentage offset of mean to the headtracked (ht) situation are given for eight different viewing conditions. The first table with columns SC and SR is for selection. The second table with columns MC, MR, MCE and MRE is for manipulation. Columns MCE and MRE indicate manipulation accuracy measurements in centimeters.

Selection	**Cursor (SC)**			**Ray-Based (SR)**		
	M(s)	SD	%	M(s)	SD	%
ht	2.0	1.16	+0	1.5	0.50	+0
not ht	2.3	1.24	+15	1.5	0.59	+3
25	2.1	0.85	+4	1.4	0.40	-3
50	2.6	1.47	+29	1.9	0.92	+30
75	3.1	1.99	+55	2.3	1.04	+55
25+w	2.0	1.16	-1	1.4	0.58	-5
50+w	2.2	1.51	+9	1.4	0.43	-6
75+w	2.0	1.04	-1	1.7	0.64	+15

Manipulation	**Cursor (MC)**			**Ray-Based (MR)**			**Cursor (MCE)**			**Ray-Based (MRE)**		
	M(s)	SD	%	M(s)	SD	%	M(cm)	SD	%	M(cm)	SD	%
ht	2.1	0.85	+0	2.4	0.76	+0	0.9	0.49	+0	1.0	0.44	+0
not ht	2.3	0.88	+7	2.5	0.97	+0	1.0	0.54	+6	1.0	0.57	-1
25	2.5	1.00	+17	2.7	0.89	+9	1.2	0.82	+26	1.1	0.62	+17
50	2.8	1.17	+33	3.0	1.06	+22	1.2	0.85	+33	1.4	0.84	+43
75	3.4	1.49	+57	3.4	1.24	+39	1.4	0.91	+48	1.8	1.19	+88
25+w	2.4	1.20	+12	2.5	0.82	+0	1.0	0.68	+9	1.0	0.53	+5
50+w	2.7	1.19	+24	2.7	1.22	+10	1.1	0.51	+15	1.2	0.64	+24
75+w	2.8	1.05	+32	2.8	1.12	+15	1.2	0.58	+31	1.2	0.66	+26

tracked situation, there is a slight increase of standard deviation. We must note that the manipulation task itself was more difficult under increased viewpoint offset conditions. The distortion would cause the images to be projected at screen edges or almost off-screen. Also the trajectories became longer and slanted as the distortions increase, making the task more difficult. In some tasks, such as ray-based selection, task performance under distorted view conditions is comparable to the normal headtracked situation. These results are in agreement with the results presented in [Simon 07].

An interesting observation is the positive effect of viewpoint compensation on direct interaction, which is essential on close-range VR systems. Under distorted viewing conditions, co-location between the physical space and the sheared-perceived scene is lost. Without correct co-location, direct interaction tasks would be almost impossible to complete without any visual feedback. We clearly see this from the large negative impact of view distortion on the cursor-based selection measurements. In this situation, the user solely relies on visual feedback of the cursor. When viewpoint compensation is applied, co-location is maintained. The experimental results indicate that, even though the world is sheared, the selection performance is brought back to the level of the normal headtracked situation.

4.6 Conclusions and Discussion

In this chapter we extend earlier work on multi-user interaction on projection based VR. We analyzed the problems of multi-user viewing and interaction on the Virtual Workbench, a close-range VR system. We first discussed possible solutions and describe the camera models and compensation techniques used to ensure visual and interaction consistency for the interacting users. We implemented these techniques in our VR development framework and performed both a quantitative and qualitative user study to evaluate them. We evaluated effects of various amounts of view distortion on task performance and accuracy, and the impact of our view compensation approach. For this, we focused on basic, close-range object selection and manipulation tasks on the Virtual Workbench.

Motion parallax is essential for a usable work experience on the Virtual Workbench. Our dynamic average viewpoint is an alternative camera model that introduces motion parallax and retains stereo for multiple users. The stereo parallax is in this case restricted to the horizontal axis to avoid overall fusion problems. Evaluations show that such an alternative camera model can improve the overall collaborative user experience. At the same time, by using the viewpoint compensation approach, usability of interaction tools can be maintained, also in direct, close-range situations. It removes confusion and discomfort during interaction in distorted spaces by making interaction consistent with the user's perception of the scene. In many cases, interaction with view compensation can be as effective as in the single-user case.

We experienced that complex rotation and docking tasks in distorted spaces are more difficult to make consistent, since rotations can change the shape of the objects. We describe our recent approach to solve this issue in [Koutek 07]. Image quality and usability can also be further improved by taking into account individual eye-separation and lighting effects [Wartell 99]. Furthermore, we plan to design a set of generic solutions and guidelines to extend existing interaction techniques. Based on our current experiments, we expect our method to enhance the effectiveness of collaborative work on the Virtual Workbench. We feel our approach is an attractive solution for many applications because it is easy to implement and accessible. We are currently extending our VR-interaction framework to provide a flexible configuration of viewing and interaction scenarios. Finally, we want to evaluate the new possibilities in real-world VR applications for collaborative data exploration and visualization.

Part II

Architectures for 3D Interaction

5

Interactive Software Prototyping for VR Applications

This chapter contains an extended version of an article published earlier. The original peer-reviewed article [de Haan 07b] was presented at the IEEE Virtual Reality Conference, Charlotte (NC), United States, 2007.

Overview

The development of domain-specific Virtual Reality applications is often a slow and laborious process. The integration of the domain-specific functionality in an interactive Virtual Environment requires close collaboration between domain expert and VR developer, as well as the integration of domain-specific data and software in a VR application. The software environment needs to support the entire development process and software life cycle, from the early stages of iterative, rapid prototyping to a final end-user application. In this chapter, we propose the use of flexible abstraction layers in the form of a dynamic scripting language, which act as the glue between VR system components and external software libraries and applications. First, we discuss the motivation and potential of our approach, after which we overview related approaches. Then, we describe the integration of a Python interpreter in our VR toolkit. The potential of our integration approach is demonstrated by its rapid prototyping features, the flexible extension of core functionality and the integration of external tool kits. We describe its current use in applications, interac-

tive demonstrations and education and conclude with an overview of implications our approach has for the future development of new framework features and application integration.

5.1 Introduction

After decades of experience with Virtual Environments, a large gap still exists between developer and end-user experience. Although the robustness of 'core' VR technology increases, other fundamental issues in VR application development arise from the typical requirements that characterize custom VR applications. These include the continuous involvement of end-users and interactivity while maintaining performance. A highly flexible software architecture is needed to match the nature of the development and maintenance life-cycle of VR applications.

During our research on interaction techniques in Virtual Environments, our VR software framework gradually evolved and extended. Recent application and framework refactoring efforts revealed that productivity is limited by the long development cycles that go into *(re-)designing, (re-)implementing and debugging* new features in our C++ software architecture. This is a limiting factor during our collaboration with domain experts, and influences the direct applicability and acceptance of VR solutions.

In general, a VR application is built on the underlying VR framework, which consists of a set of carefully orchestrated heterogeneous components. Especially in the early, exploring stages of collaboration with domain-experts, many changes continuously evolve the ideas on and the state of the VR application. The provided framework API needs to be flexible enough to enable use and extensibility, while avoiding a bloated, overall up-front design. In addition, there is also the need for expressive end-user modeling and programming. Therefore, we opt for a continuous, rapid software prototyping environment, in which many small features and links between components can be tested and evaluated. A loosely designed set of high level abstractions, reusable components and code snippets allow developers and eventually experienced users to use rapidly create complete applications by extending basic functionality in an ad-hoc fashion.

We propose a VR application development paradigm based on flexible abstraction layers through a single abstraction language. In our approach, the abstraction layers are code fragments that abstract, combine and wrap lower level code. The goal of this approach is to facilitate *(1)* continuous, iterative software development, including features such as rapid prototyping, profiling and debugging, *(2)* flexible integration and configuration of heterogeneous VR and external software, *(3)* seamless evolution from early software prototypes to flexible end-user applications and *(4)* ease-of-use, lowering the learning curve and empowering end-users. In other words, already early in the development process developers generate an easy-to-use, application specific API and VR elements by simply glueing together several lower

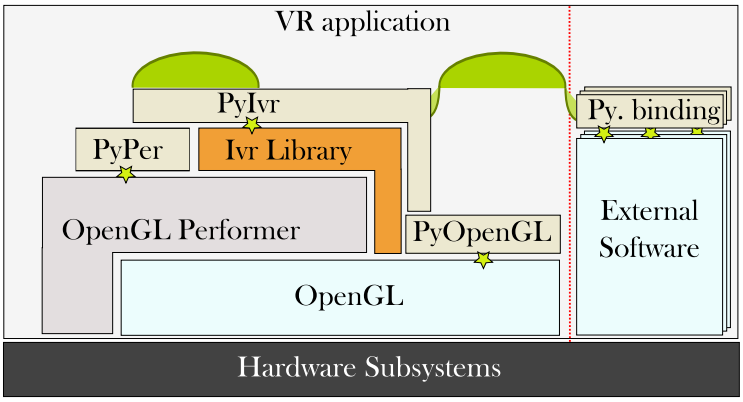


Figure 5.1: Overview of the iVR software layers (see Section 5.3.1). The VR application has native access to various components and external software or through Python. Stars indicate Python bindings on the underlying libraries. Arches indicate custom Python layers which provide higher level abstractions.

level VR and application specific elements. The flexibility of the glueing language and development environment ensures an easy creation of these intermediate layers and a smooth transition from early development APIs and application prototypes to flexible end-user VR applications.

In this chapter, we describe our work towards the proposed VR application development paradigm of integrated rapid software prototyping. Here, the focus is on the flexible abstraction layers which connect several aspects of our VR library, applications and external, domain-specific tool kits and libraries. We first describe related work in Section 5.2. Section 5.3 and 5.4 describe our prototype and several examples. We conclude with a short discussion and our view on future VR application development in Section 5.5.

5.2 Related Work

Work on flexibility in application development and VR integration can be found in VR-related approaches, as well as in more general and domain-specific learning and problem solving environments. Often, approaches are separated in system-specific facilities and end-user application specific features.

In *VR system frameworks*, run-time flexibility —on a system level— has been acknowledged and applied many times. The Bamboo [Watsen 98] plug-in framework is an early system providing a micro kernel-based VR architecture in which all system elements are pluggable components that could be added, removed, and replaced at run time. In VR Juggler [Bierbaum 01] the possibilities of run-time reconfiguration of the VR system to a higher level than only the system are described. The focus in

these approaches is mainly on system configuration flexibility, and less on flexibility of behavioral aspects of specific VR application. In VHD++ [Ponder 03], also the integration of domain-specific applications in such a framework is discussed, while maintaining extensibility and flexibility. The approaches mentioned above achieve their flexibility through existing and well documented software design patterns for the creation of a solid system framework and protocols.

Dynamic scripting languages enable iterative design techniques, allowing a dynamic approach to application and system design. Many VR systems provide these facilities in their frameworks to achieve development flexibility. As part of this work, we maintain an online overview [1] of VR frameworks and graphics APIs in which scripting support is provided. In our prototype description we discuss scripting language features and their integration in more detail. In all listed approaches, with Visum [Finkenzeller 03] being a notable exception, a C/C++ based VR kernel and graphics API is used. Visum uses Python and PyOpenGL as the core of the framework. The level of abstractions and scripting integration features differs widely between systems. The common aspect of all approaches is the availability of high level abstraction layers on top of the complete VR framework. In addition, Coloseum3D [Backman 05], Panda3D [Goslin 04] and Avango [Springer 00] also use a scripting language as the main integration layer of various system components, providing low-level, flexible access to developers and end-users.

Here, *specification of interactivity* is also main aspect of VR application development. An early example of 3D graphics interaction specification in a native scripting language can be found in the Alice toolkit [Conway 97]. An analysis of describing and modeling interaction behavior is given in [Burrows 05]. Modeling languages are used to avoid programming and validity issues for the end-user, while maintaining flexibility. This can quickly become complex, as describing behavior beyond simple examples requires more elaborate constructs (such as control-flow elements) and one needs to interact with application code, on a VR system-level, but maybe also on a domain-specific level through external software. Hendricks et al [Hendricks 03] propose an interactive, scripting approach to overcome many issues of this duality in modeling language. We want to provide the user with a unified syntax and semantics for describing interactivity with (external) application components.

In many *domain-specific application areas* rapid prototyping facilities were proposed, ranging from high level APIs and Domain Specific Languages to full-blown Problem Solving Environments. On many levels, scripting support is available. The *tinkering* during experimentation and development proves useful [Ousterhout 98] and is finding its way into new software design methods such as extreme programming and aspect-oriented design. SuperGlue [Hultquist 92] uses Scheme language constructs for describing a visualization process of domain-specific data. It is presented as an answer to the existing visualization platforms that overemphasized ease-of-use by the use of GUI, which failed to adequately address issues of exten-

[1] http://visualisation.tudelft.nl/VRdev

sibility. VHD++ [Ponder 03] uses stub components to which application developers should connect their applications. Here, flexibility in this approach is achieved by a strict communication protocol and design pattern.

5.3 Prototype Description

The main aspect of our proposed VR application development paradigm is the use of abstraction layers, on which all interactive, programmable aspects of the VR application and individual components are founded. Our current prototype implementation is a first step towards the application of these abstraction layers in a VR application development environment. Currently, the prototype is mainly intended for experienced VR developers, but some high level features geared towards end-users are also demonstrated. Furthermore, new application-specific high level facilities can quickly be developed based on the new abstraction layers.

5.3.1 Software Layers

A schematic overview of our *interactive Virtual Reality (iVR)* system is shown in Figure 5.1. This is a recent rewrite of our *RWB library* [Koutek 03], a closed, C++, OpenGL Performer based VR toolkit, where its monolithic characteristics were transformed to a micro-kernel approach, augmented with separate flexible components. The set of components and domain-specific applications, combined with the scripting language layer, now form the basis for creating a VR application.

We selected Python as the unifying abstraction language layer for multiple reasons. First, this General Purpose Language integrates well with existing C and C++ code. The Python interpreter can be extended or the interpreter can be embedded in the C application. Second, a solid basis is provided by its wide availability on various platforms, a large standard set of tools, and wrappings of many (scientific) software packages. Its simplicity and flexibility allow for a smooth transition from powerful lower-level access to higher-level, more user-friendly constructs. Also, the interactive introspection and self-parsing facilities enable us to extend the language with special purpose (such as domain-specific) sub-languages.

5.3.2 Wrapping of existing software components

Abstraction layers in the high level scripting language form the basis for interactive control of various systems components. Special conversion code is necessary to take care of the type and value conversions between the C++ and Python environments. This intermediate conversion code layer, the so called wrappers or binding, can be created manually or (semi-) automated by external software. The level of automation is determined by the wrapping method used and the complexity of the C++ constructs.

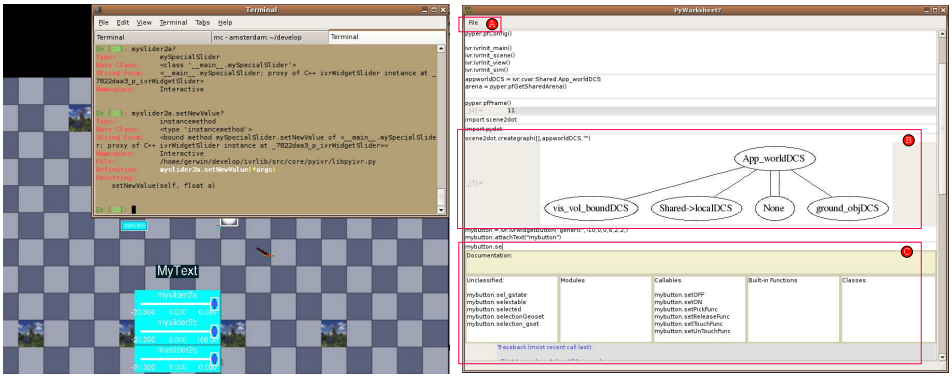


Figure 5.2: The interactive Python shell (left) and the notebook editor (right). The interactive shell can be used while the VR application is running. The IPython shell provides interactive script access to the running VR application. Code completion, documentation functionality and many more functions ease development. The notebook or worksheet (right) shows code editing, loading/saving operations (A), integrated graphics (B), and available documentation and command completed parameters (C). The interactive graphics or controls in (B) are generated by customizable Python helper code.

Many Python wrapping generators exist, of which *SWIG*[2] and *Boost.Python*[3] are the most popular. SWIG parses declarations in header files to generate a intermediate Python and library file, which expose the wrappings [Beazley 03]. Boost.Python uses *template meta programming*, and uses some helper code and the C++ compiler itself to generate a wrapper library. Difficulties can arise when complex data is communicated between two C++ components for which different wrapping generators are used. Other issues in performance and usability also influence the choice of the wrapping solution, but technical details are outside the scope of this document.

Our iVR toolkit intensively uses OpenGL Performer functionality through *PyPer* [Stolk 02], SWIG generated OpenGL Performer bindings. We use SWIG to wrap our iVR library to avoid cross-wrapper difficulties. In this way, we are safe to transparently mix Performer data types and functions with iVR functions in a Python environment. We use two helpful SWIG provided features to enhance the usability of our wrappings. First, SWIG *Director* classes enable *cross language polymorphism*, allowing for easy subclasssing and extension of existing C++ classes in Python. Second, source code documentation such as comments, parameter and data types are made directly available in the Python interpreter. Using our wrappings, we obtain most functionality directly in Python syntax. The thin and adaptable layering of abstraction levels can provide both flexibility and performance on a low-level, while on a higher level the ease-of-use and expressiveness is maintained.

[2] http://www.swig.org/
[3] http://www.boost.org/libs/python/doc/

5.3.3 Control Beyond Wrapping

The structure and functionality of the generated wrappings do not always match the requirements for interactive, run-time development. First, a greater error robustness is required in both the wrappings and the underlying software. Validity assertions and good error handling, including documented error reports is necessary for the interactive development. Second, the directly mapped language constructs may not be on the right abstraction level for the task at hand, for both the developer or end-user. The advanced programming styles and thin abstraction layers in Python snippets enable the use of more flexible software construction. Third, the control flow from different software components must be combined. For example, we need to combine the Python interpreter control, the OpenGL Performer render loop, and even a physics engine. In our method we opted for *extending* Python, which makes the Python interpreter the overall controller of the VR application. This provides us with many features to inspect system state and manipulate the control flow. Finally, performance can be problematic if many operations need to be performed in the Python interpreter. Normally, as much of the code will be glue code and initial set-up code, the performance bottleneck of the VR application will not be in the Python handling. For improved performance one can resort to native C++ components and make use of multi threading and multi processing.

5.4 Prototype Results

This section demonstrates the important features of the flexible abstraction layers through sample applications. A VR application can be completely written in a script, which is executed by a standard Python interpreter. The iVR functionality is directly available in the running Python interpreter after importing its wrapping modules.

5.4.1 Run-time Prototyping

The Python interactive interpreter allows interactive control in created scripts. It facilitates a run-time prototyping environment, because program functionality can be inspected, added and changed. We use *IPython*, an enhanced interactive interpreter to improve the usability and interactivity of this process. IPython provides many extra development features, including object and code inspection, command history, integrated debugging and saving of interactive prototyping sessions, see Figure 5.2 (left). Both the regular and IPython interactive interpreter are *line-based* concepts, where entered sets of lines are interpreted and executed immediately. The use of small, saved testing scripts leads again to a differentiation between saved code and interactive code. We use experimental software developed in our group for the construction of Python-based VR code using the Notebook metaphor. This metaphor provides a unified worksheet for the code, interleaved with interactive graphics, see

Figure 5.2(right). It provides a smooth transition from interactive prototyping to working code snippets. Graphical representations or interactive widgets are created by the use of small helper snippets, which generate the content based on special object types.

5.4.2 Internal Extensions

The iVR toolkit, with its abstraction layers and a set of standard Python snippets, provides an accessible, high level VR application skeleton from which development can start immediately. The user can construct his application by using and extending a set of standard widgets, graphical objects and interaction techniques. The close integration of Python and the original C++ code allows us to, gradually, transform existing code toward Python oriented programming methods. These extensions can range from simple widgets, such as a simple valuator with custom behavior (see Figure 5.3), to complete custom interaction handling mechanisms. Our recent Python-based iVR developments illustrate the flexibility. We developed an event mechanism and state machine components for interaction and behavior modeling. We use *state charts*, a technique for modeling the system state with concurrent, hierarchical state machines. We combine run-time code inspection and interactive state chart manipulation, providing a valuable insight during the interaction development cycles. While the VR application is running, state charts and transitions can be extended and have their graphical representations generated. These new facilities again form new abstraction layers that are easily extended. The use of other external libraries and toolkits allows to further enhance the development process. For example, one can think of GUI-based interaction modeler on top of the current state chart abstraction layer.

5.4.3 External Software Integration

When integrating external software in VR application constructs, the use of the abstraction layer shifts many difficulties to the wrapping generation process. Once wrappings are available, or if software is already in the abstraction language, constructions from the various components can be mixed with less effort. The limita-

```
class mySlider(ivr.ivrWidgetSlider):
 def SliderExtensionFunc(self):
     print "Value=",self.getValue()

myslidero = mySlider("Iso-value",worldDCS)
```

Figure 5.3: Extending Functionality in Python. This code snippet shows a new Python class derived from a C++ class, with a custom callback attached. Then an instance is created. Although valuator handling is in C++, the Python callback will be used.

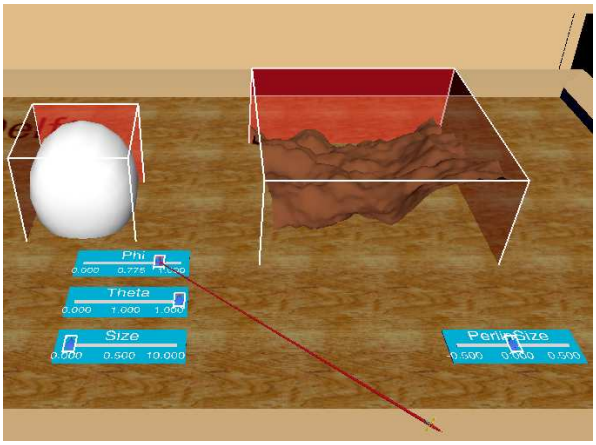


Figure 5.4: Demonstration of bi-directional integration of VTK in the VR environment. Graphical data from two VTK pipelines is shown in the VR application. Callbacks from widgets directly control parameters of the two VTK pipelines. The Python glue facilitates expressive commands that combine VTK and VR statements

tions of this mixing are dependent on the data size and data format compatibility between the various components.

External software of special interest are domain specific applications and libraries and general-purpose functionality that can be useful for analysis during development. In our current prototype, we integrated *matplotlib*[4], a Python package for mathematical plotting, *Graphviz* [Ellson 03], graph construction software, and the *Visualisation ToolKit(VTK)* [Schroeder 06], a data visualization package. The use of Graphviz is demonstrated in a previous Figure 5.2, where graphs are integrated in the Notebook environment. This integration with the VR application is currently *uni-directional*. This means that results of Graphviz functions on data structures such as scene graph and state machine hierarchy, are shown in the Notebook environment only, and not directly back in the VE.

For VTK integration in our VR software we wrap the *vtkActorToPF* library to do performance critical conversions from VTK to Performer data, and provide a mixed iVR-VTK abstraction layer. The end-user can mix VTK code directly with iVR constructions in Python scripts, while the abstractions layers perform the underlying communication between the external libraries. VTK generated graphical objects are created by using VTK commands directly in the VR script or by importing an external VTK example file. Figure 5.4 illustrates this *bi-directional* approach, where resulting objects are first-class objects in the VR application and can be directly integrated with VR interaction and behavior.

[4]http://matplotlib.sourceforge.net

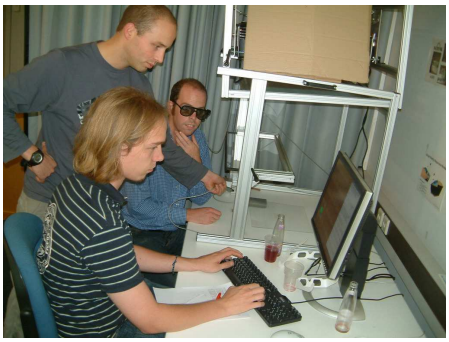



Figure 5.5: Interactive application prototyping for education. During a Virtual Reality workshop for PhD students, our prototyping approach was used for interactive demonstrations (left) and hands-on work on various VR systems (right).

The construction of real, domain-specific VR applications builds upon the basic application skeleton using the integration and extension techniques described above. As the entire work flow is an extensive and iterative process, a running VR prototype can be constructed and maintained throughout the cycles of development. As wrappings and abstraction layers for the core functionality of the external software packages are introduced, they can gradually be connected with the VR components. During this process, interactive prototyping experiments with domain-experts and VR developers can lead to insights on new application requirements, for example the need for specialized visualization and interaction techniques.

5.4.4 Iterative Development

As described in earlier sections, we use the interactive prototyping for the development of new features. During recent usability studies in Virtual Reality, the interactive development features and the ability to integrate external functionality proved valuable. The entire, gradual process involved designing and setting up the experiments, performing timing and file operations to the graphical and statistical analysis. Here, we made extensive use of the interactive scripting capabilities for interactive experiments and the integration of external libraries. As the data stored in the acquisition process could directly be interpreted with graphing and statistical libraries, we learned that this integrated approached provided enhanced productivity over the use of several external software packages.

The iterative development features proved valuable for educational purposes as well. During the 2006 ASCI PhD course on Visualisation and Virtual Reality (a17), over 25 PhD students worked with the VR hardware in our labs. The iVR library and its interactive scripting mode was used throughout the workshop for both demonstrations and hands-on work, see Figure 5.5. After a short, interactive demonstration

on the system the students worked for a day on a series of assignments ranging from programming simple interaction and widgets to the integration of a medical dataset in VR using VTK. This experience indicated that the learning curve of the interactive, script based version of our iVR library has lowered significantly compared to our original C++ based version.

5.5 Conclusions and Future Work

The introduction of multiple abstraction layers in existing VR software tool chains using dynamic languages such as Python provides flexible development styles for VR application development. The ease of programming and the multitude of abstraction layers allow both developers and end users to use expressive programming commands at a suitable level of comprehension. A powerful functionality included in Python and the availability of wrappings for external software packages ease the process of application integration. We described the features and benefits of the abstraction layers through the gradual introduction of a Python layer in our existing, C++ based VR toolkit. The interactive scripting environments give run-time access to the running application, providing an interactive prototyping environment. As stated earlier, development efforts are shifted towards the creation of general library wrappings as well as interactive development environments. For interactive use, greater error robustness is required in both the wrappings and the underlying software, as well as useful debugging information.

It must be noted that the approach described does not provide a general-purpose and ready-to-use API for VR end-users directly. Furthermore, it does not solve the integration of VR and application tool kits in general. Instead, the development effort put into these issues, on a per-application basis, shifts towards the creation of simple and flexible APIs. If done early in the development process, these APIs combined with availability of the general run-time development (VR) environments can accelerate the creation of effective VR applications.

We are working towards a rapid VR prototyping paradigm that provides a solid base for various development styles. The transformation towards interactive control through unified abstraction layers catalyzes the re-design of previous software mechanisms and a change in development philosophy. We envision an integrated development and run-time environment providing interactive control using higher level, visual programming and debugging tools. We expect the abstraction layering and integration of external tools to be key aspects in achieving this goal.

6

StateStream: a Developer-Centric Approach Towards Unifying Interaction Models and Architecture

This chapter contains a slightly modified version of an article published earlier. An early version of this work in progress was presented at the SEARIS workshop [de Haan 08a]. The final, extended version of this peer-reviewed article [de Haan 09a], was presented at the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS) in Pittsburgh (PA), United States, 2009.

Overview

Complex and dynamic interaction behaviors in applications such as Virtual Reality (VR) systems are difficult to design and develop. Reasons for this include the complexity and limitations in specification models and their integration with the underlying architecture, and lack of supporting development tools. In this chapter we present our *StateStream* approach, which uses a dynamic programming language to bridge the gap between the behavioral model descriptions, the underlying VR architecture and customized development tools. Whereas the dynamic language allows full flexibility, the interaction model adds explicit structures for interactive behavior. A dual modeling mechanism is used to capture both discrete and continuous interaction behavior. The models are described and executed in the dynamic language

itself, unifying the description of interaction, its execution and the connection with external software components.

We will highlight the main features of StateStream, and illustrate how the tight integration of interaction model and architecture enables a flexible and open-ended development environment. We will demonstrate the use of StateStream in a prototype system for studying and adapting complex 3D interaction techniques for VR.

6.1 Introduction

The look and feel of a well designed interaction technique appears simple, logical and intuitive to its users. Little thought goes to the often painstaking and time-consuming development process of making interaction techniques work as designed, error-free and well-tuned. Even with use of existing interface and interaction modeling methodologies, taxonomies and software tools, it remains hard to design, to model, to integrate, to debug and to evaluate complex interaction techniques and scenarios. Trends in new input modalities such as multi-user displays and distributed systems further complicate the design and development of interfaces, the underlying software models, architectures and tools.

In our recent efforts in developing multi-user and multi-handed input 3D interaction techniques [de Haan 07c], we encounter many situations where the complexity of the interaction description explodes. Simple interaction techniques are relatively straightforward to design and implement, but their re-use through variations and combinations can easily lead to unexpected results, often only to be discovered while already applied in a VR application.

Consider the example in Figure 6.1, where a 3D box can be selected and freely manipulated with a ray controlled by a tracked stylus. This works intuitively for a single user operating the VR system, but when a second user joins in and grabs the same object, the interaction behavior can become more complex. As a side-effect of our original event-based implementation, the object's control is just taken over by the last selecting user. Naturally, one would want to decide which alternative behavior is used, such as averaging interacting forces, bending one user's ray or scaling the object. The relations between all components depend on which combination of interaction tools operate on which object types. To allow for these alternatives, one would need to rethink and re-implement the interaction behavior in detail.

Although much effort can be put into capturing these new situations in hard coded, imperative commands of callbacks and event-handlers, problems concerning the number of object relations and *exceptions to the rule* either get overlooked or quickly overwhelm development tasks. In practice, classic interaction modeling concepts and development tools often lack integration and rarely provide the right level of abstraction for effective design and problem solving. These restrictions often prevent developers and designers from adopting and customizing more sophisticated 3D interaction techniques in real VR applications.

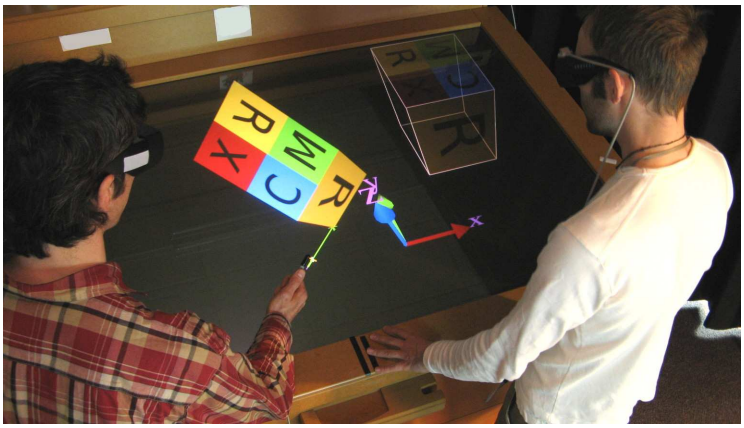


Figure 6.1: Example of a dynamic, 3D manipulation technique in a multi-user VR setup. When multiple interaction tools can operate on multiple objects, interaction behavior can become difficult to design and program.

Our motivation for this work follows from these issues. We feel that (combinations of) interactive behavior are inherently complex and require model-based design and supporting analysis tools. At the same time however, one wants to avoid restrictions a model imposes on the flexibility of existing software tools and existing design skills. A flexible integration of model, architecture and the supporting tools is important to support developers with varying skills and backgrounds, ranging from graphics programmers to interaction designers. The main problem is that many existing interaction models are far separated from other, external software components. This separation limits their descriptive power, thereby restricting the visibility of features and issues when integrated in run-time environments. This makes it difficult to appreciate a model's value, especially in agile scenarios with many software components and a cyclic process of design and development, see Figure 6.2. In this chapter, we address this issue of separation and discuss our developer-centric approach. We present *StateStream*, a pragmatic software approach to unify interaction models and architecture.

The contribution of StateStream is the developer-centric approach of using a dynamic language to unify an interaction model with underlying architecture and tools of interactive applications. The interaction model provides semantic structure, but is described using in the same language with familiar syntax as the other system components. It allows integration and transition of existing code and control structures, but also benefit from dynamic language features such as dynamic execution, introspection and extension at run-time. With this approach we have implemented and integrated a dual interaction model, consisting of separated StateChart and Data Flow primitives. We describe how this model provides powerful composition pat-

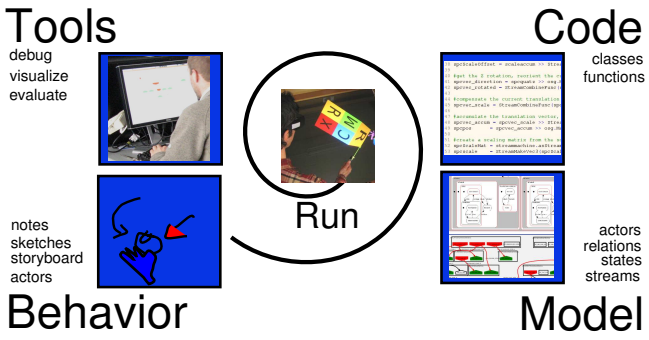


Figure 6.2: Cyclic development. From an informal behavior description, one generates a model abstraction which closely maps to running code. Errors or unexpected situations may occur during interaction, and need to be detected, analyzed and fixed. StateStream integrates model, code and tools.

terns for code-reuse, how it eases integration with underlying system components. To demonstrate the functionality of our model and approach, we describe the creation, adaptation and re-use of several interaction tools within our VR framework.

The remainder of this chapter is organized as follows: We first discuss related work on interaction models, tools and architectures of current interactive graphics systems in section 6.2. Then, in section 6.3, we describe the StateStream model and its components in technical detail. After a description of the implementation in section 6.4, the process of creating several 3D interaction techniques within our VR demonstrator is described in more detail in section 6.5. Finally, we discuss the results of this work in section 6.6 and conclude and give our view on future work in section 6.7.

6.2 Related Work

In this section we discuss the position of the StateStream approach with respect to related work. We consider three main themes of interest: model-based design, its practical integration in the underlying architecture, and the supporting software tools in the development cycle.

6.2.1 Model-based Design

Modeling languages are considered an essential asset in describing and implementing interaction behavior while avoiding detailed execution and validity issues for users. Among the various models, the Data Flow paradigm and state-based models are the most well-known for user interaction. The Data Flow model is widely applied to describe system flow and interaction techniques in terms of filters, often with a focus on reconfiguration. For example, with the InTml [Figueroa 02] specifi-

cation language one can describe 3D interaction techniques, input devices and their connections. UNIT [Olwal 04] uses a similar model and focuses on flexible redefinition of continuous behavior of interaction techniques. The IFFI system [Ray 07] provides an even higher abstraction to allow for reuse of techniques across different VR toolkits. FlowVR [Allard 05] extends the Data Flow approach over its entire VR architecture to provide distributed capabilities. Data Flow models excel in their description of continuous behavior components, but often require ill-formed constructions to support simple tasks such as message passing and event handling.

State-based models are better suited to model discrete behavior and integrate with event-based systems. For specifying reactive user interface behavior, special interest goes to StateCharts [Harel 87], which provides a visual formalism for hierarchical, concurrent statemachines. The hierarchy and concurrency can avoid state explosion, see [Wingrave 08]. Many StateChart variants exist with different model properties and operational semantics [Beeck 94, Lüttgen 00]. Recent examples of StateChart-inspired user interface modeling approaches include *HsmTk* [Blanch 06] for 2D direct manipulation interfaces, *CHASM* [Wingrave 05] for 3D user interfaces and *d.tools* [Hartmann 06] for physical prototyping. The *SwingStates* system extends the Java Swing user interface toolkit with concurrent statemachines [Appert 06].

At discussions during the IEEE VR 2008 SEARIS workshop (Software Engineering for Realtime Interactive Systems, [Latoschik 08]), participants acknowledged that the use of key aspects of multiple modeling techniques is a promising approach. We observe that this integration does exist in many approaches to some extent, but often lack an explicit separation between these two models. This quickly introduces complex *feature interaction*, which complicates application behavior analysis. Similar design issues were already addressed earlier in the specification of interactive systems. For example, a separation of *status* and *events* phenomena to model state and continuous relationships between interface components is used, for a recent implementation see Dix et al. [Dix 08]. An alternative approach to specify user interaction in the context of Virtual Environments is proposed by Smith et al. [Massink 99, Smith 07]. They extend high-level Petri Nets with Data Flow, so-called *FlowNets*, and use a semi-formal notation to model discrete and continuous components. This work is requirements-centric as it aims to provide only a *sketch* of interaction to enable analysis of usability requirements.

In contrast, in our developer-centric approach we emphasize unification of model and architecture for many development iterations. The familiarity of developers with models and their mapping to underlying language and system structures is important. Therefore, we chose to mix state-based models and Data Flow to model event-based and continuous interaction. An early example of this model separation is shown in [Carr 94], where StateChart diagrams are expanded with Data Flow and constraint specifications to design custom user interface widgets. The use of this type of model separation in real-time interactive graphics systems is demonstrated in the HCSM driving simulator [Cremer 95]. Jacob et al. transfer this approach to the de-

scription and programming of non-WIMP user interfaces [Jacob 99, Shaer 05]. Our StateStream system builds on a similar, dual-modeling approach where StateChart mechanisms and Data Flow are separate, first-class primitives. The main difference of our work with the systems above is the way the models are described and implemented.

6.2.2 Model Integration

Model-based techniques are of practical relevance if results can be effectively integrated with the underlying software architecture, such as a scene graph system. From a historical perspective, Myers [Myers 00] reports that both formal language based tools and model-based techniques for UIMS's suffered from a high *threshold* of acceptance. In many model descriptions, a specialized description language is compiled or interpreted to running code. For example, XML-based descriptions, sometimes augmented with native code snippets, are first converted to C-style code, then compiled and run, see e.g. [Figueroa 02, Vanacken 06, Dix 08, Wingrave 08]. However, Carr already stated the potential of editing specifications and directly executing them [Carr 94].

We consider the semantics and syntax of the modeling language to be determining in this high threshold. First, declarative description model semantics may restrict expressiveness, especially with respect to the pure imperative coding practice on underlying architecture. Second, language syntax is often different. Third, once code is compiled and run, the relation between the running code and the mixed description and code is difficult to grasp. As a result, conceptually related elements may exist in different language semantics, syntax, files and scope, which make development and debugging difficult.

To alleviate some of these issues, we avoid the use of a specialized declarative model language. Instead, we explicitly describe interaction techniques using the constructs in Python, a dynamic, easy-to-learn programming language which unifies our framework. Zachmann proposed the use of a special-purpose scripting language on a fixed VR interaction API [Zachmann 96]. Early versions of *Alice toolkit* provided a Python-based scripting environment for interaction and low-level extensions to lower the learning barrier [Conway 97]. Although the use of a dynamic language by itself does not provide a ready solution, its flexible syntax and interpreter allow a clean integration of *pseudo-declarative* models. The usefulness of a similar integration approach is demonstrated in the *SwingStates* toolkit, where statemachines are described in native Java inner classes [Appert 06].

We chose Python because the dynamic language offers more flexibility, such as introspection, for building development tools. Also, the integration with a scene graph system builds upon earlier work on flexible abstraction layers of our base VR architecture [de Haan 07b]. The StateStream model is self-executing and operates at run-time in Python, allowing interaction techniques and scenarios to be studied and modified, often without restarting the application.

6.2.3 Development Environment

Finally, we consider the development environment and run-time system to be an essential element of interaction design software. Wingrave et al. [Wingrave 08] report on the complexity of interface development and argue the need for better, developer-centric methodology and tool support through the entire design and development cycle. Hendricks [Hendricks 03] highlights the need for VR interaction to allow a smooth migration of novice users to becoming more experienced and proposed the use of *meta-authoring tools* which assist in the creation of interactions. Some of these ideas appear in the *d.tools* environment [Hartmann 06], which provides an integrated design, test and analysis environment. However, like *NiMMit* [Vanacken 06], d.tools is a strongly visually-oriented design tool from which code is generated and run. This visual approach generally does not scale well to larger systems and restricts low-level inspection or modification of application states and flow of control by more experienced designers and developers.

Our approach focuses on the low-level, fine-grained aspects of 3D interaction and detailed model behavior, and less on an Integrated Development Environment (IDE) for end users. Because of this low-level approach, the heart of our approach is an accessible, programming language-based interaction description. This description can be run directly, while components for analysis and visualization such as graphs, traces and lists for the front-end GUI communicate with the running model. As a result, the dynamic language serves as a unifying link for describing interactivity between objects, connect application components, and to produce front-end development and analysis tools. We feel this integration helps to lower the barriers between different stages of development from design, programming and analysis. With this approach, we attempt to reduce system viscosity by providing higher *flexibility* and *expressiveness*, necessary to rapidly iterate to a better user interface system, as suggested by Olsen [Olsen 07].

6.3 Model Description

In this section, an overview is given of the StateStream *model primitives* and how they integrate *domains* of behavior description. As described in the previous section, a dual modeling approach is used, similar to PMIW [Jacob 99]. The first modeling primitive is the *statemachine*, a StateChart-like mechanism intended for describing behavior in the *discrete domain*. The second modeling primitive is the *streammachine*, intended for modeling conceptually continuous streams of information, thus in the *continuous domain*. A third domain is the *actor domain*, which essentially forms an interface to the underlying architecture and contains the statemachines and streammachines. Figure 6.3 gives an overview of the various components and the control mechanisms through which they influence each other. Before we describe these primitives in more detail, we first motivate the choice of description language for

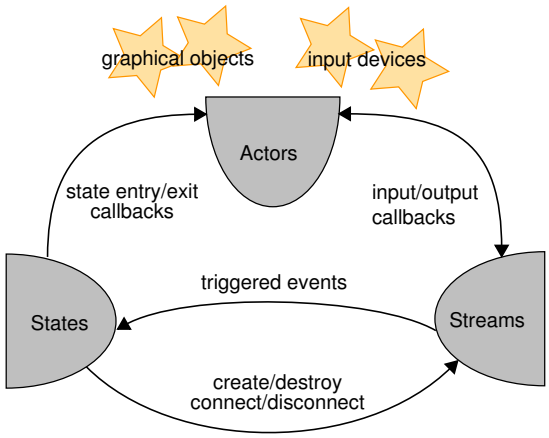


Figure 6.3: Three main StateStream components. Arrows indicate the main mechanisms through which components influence each other.

model primitives and application description, which is essential in our developer-centric approach.

6.3.1 Description Language

One key aspect of our approach is that the StateStream model primitives are described directly as class structures in a dynamic, interpreted language, i.e. Python in our case. This avoids the use of a specialized model description language, parser and compiler or interpreter. Object-oriented techniques such as class hierarchy and method overloading are well-known and extremely useful for composition and inheritance of StateStream model primitives. It allows a transparent communication between domains, as well as integration with functionality of the underlying Python application. Regular Python syntax is easy-to-read, and StateStream's syntax slightly extends this to have a descriptive rather than imperative appearance, although its semantics are not formally declarative, see Figure 6.4 and 6.5. In contrast to many other description languages, StateStream primitives are self-executing within a simple execution engine in a standard Python interpreter. This means they can be run and tested stand-alone, or be integrated within external Python programs, classes and libraries. The internal StateStream models and execution flow can be inspected and adapted at run-time to allow for dynamic model behavior. The main aspect of this is that new model primitives can be created, loaded and inserted during an application, which is essential when live prototyping applications or when the scene is not known in advance.

6.3.2 Actor Domain

Behavioral functionality is split into conceptual *actors*, each of which contain the two behavioral primitives as described above. In the context of a VR application, an actor often "is" or consists of a visible VR object and its behavior specification. For interaction techniques, actors include the graphical elements such as cursors, rays and information labels. Some actors do not have a graphical representation, but instead represent a proxy for example an external algorithm or interaction device. Although it is not technically restricted, actors preferably use streams and events instead of direct references to other StateStream modeled objects, nor do they contain behavior-specific logic. This is to prevent for unexpected feature interaction outside the model logic.

6.3.3 Discrete Domain

For describing discrete actor behavior, we use a simplistic *StateChart* variant, which consists of hierarchical, concurrent statemachines. A graphical representation, automatically generated from an instance of such a statemachine, is shown in Figure 6.6. In this Figure, each rounded box represents a single state. An hierarchical state can have children, which in turn can be a statemachine. Simple states allow only one child state to be active at the same time (red outlines), while concurrent states (grey, filled) have all their states active. Template states (yellow, filled) represent concurrent states that are dynamically generated and replicated. Each state can have transitions to another state. State transitions can occur if certain *events* match the conditions and filters of available transition. When a transition among hierarchies occurs, this can cause a cascade of state entries and exits. Custom functionality is defined in callback functions such as state entry and exit functions, or at transitions. These functions can be part of the actor, see Figure 6.4. For consistency, it is advisable to set properties or perform actions in a state *entry* function which can be reset or undone in the respective state *exit* function. A main functional element here is the broadcasting of events to other objects and to set state-sensitive properties of the related actor. In this domain, the flow of control of the actors and their states can be clearly modeled and visualized.

6.3.4 Continuous Domain

The *streammachine* primitive is intended for modeling conceptually continuous streams of information. For this we use a simple Data Flow graph structure, which consists of a set of connected nodes or filters with various input and output ports. The ports of the nodes can be connected through *connection* objects, over which information of various data types can be transported. The custom functionality of the nodes is again defined in callback functions on the incoming and outgoing ports. Simple Python syntax and constructions are used for creating a new streammachine class

```
#Actor reacts to "Touch" and "unTouch" events
class TouchyActor(ssActor):
    def initActor(self):
        #initialize actor, states, streams
        ssActor.initActor (self)
        self.st_NotTouching.addTransition(self.st_Touching,
                                          "Touch")
        self.st_Touching.addTransition(self.st_NotTouching,
                                       "unTouch")
        self.conn = StreamConnection()
    def st_NotTouching(self):
        #NotTouching State will be created,this is enterfunc
        self.conn.disconnect()
    def st_Touching(self):
        #connect stream to guiPrinter, broadcast event
        self.conn = self.portout_position >> guiPrinter
        addSSEvent("Hurts",self)
    def portout_position(self):
        #stream callback
        return self.getPosition()
```

Figure 6.4: Sample Python code for an actor containing states, transitions and connections. With simple syntax conventions and introspection we can create objects implicitly for states, ports, connections etc. at run-time.

```
#overloaded streammachine for widget connection
class StreamGTKValue(StreamMachine):
    def __init__(self,pwidget):
        StreamMachine.__init__(self)
        self.mwidget = pwidget
        #create an explicit port
        self.mPort_in = self.createInputPort \
                        ("IN",self.updateIn)
    def updateIn(self,pValue):
        self.mwidget.set_value(pValue)
    def portout_myvalue(self):
        #implicit output port "myvalue" from function name
        return self.mwidget.get_value()

#connect mySlider widget to a scaling function
StreamGTKValue(mySlider) >> sceneObject.setScale
```

Figure 6.5: Sample code for making a streammachine class for a GTK widgets. We use the >> operator to connect streams to ports or directly to variables or functions.

and connections, see Figure 6.5. A part of a Data Flow network is shown in Figure 6.6, this is generated by introspecting run-time objects. Continuous variables of an actor and its graphical objects, e.g. position, rotation, color or size are easily modeled and connected through filters. In a standard VR scenario, updates are typically executed every render frame and values are pushed through the network, but more advanced and optimized update strategies on the graph are possible.

One can clearly see how actor properties are related through explicit connections, when in a certain state of an application. This separation of concerns allows one to better reason on intended behavior and analyze its implementation. Although the streams are conceptually continuous, in implementations they are sampled and separated in function calls or events. The separation of the discrete domain also has an important practical implication. We do not *pollute* the discrete event system with often-called "update" events, which do not contain any state information. Especially

when inspecting event streams in a VR application with a rendering frame rate of 60Hz and more, the amount of event information would be simply overwhelming.

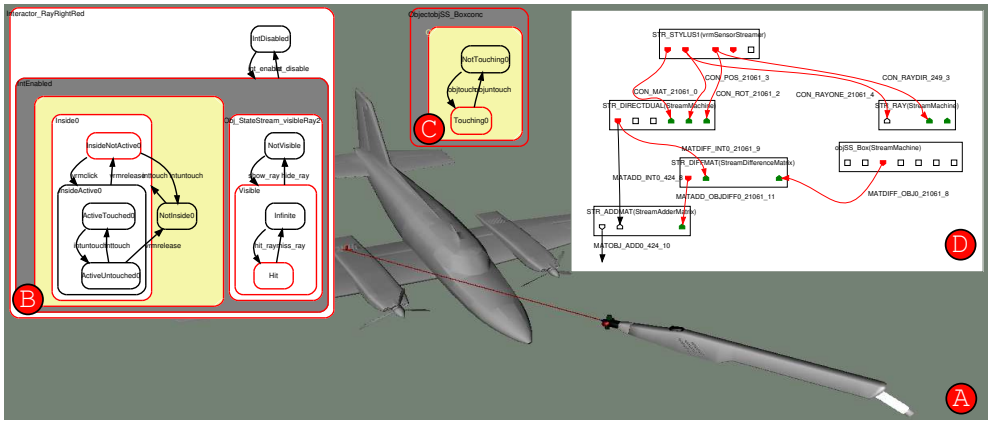


Figure 6.6: Composition of a StateStream modelled 3D ray casting selection technique, see also section 6.5.1. A stylus device can be used to point a ray at objects in the VR scene (a). Details of the modeling primitives can be found in section 6.3. Generated graphical StateChart representations for the statemachines are shown for both the selection technique (b) actor and the plane (c) actor. The statemachine of the ray actor is contained in (b, right sub-state). The related Data Flow graph (d) displays the current streammachines, their ports and connections that are in use for this state.

6.3.5 Integration

A StateStream-based application consists of the description of the actors, their statemachines and streammachines. For a StateStream application, a main, top-level statemachine is maintained which reflects the application state. On application startup, main actors are instantiated, which in turn can activate their own actors. During actor initialization, both its statemachine and streammachine structures are created. All statemachines existing in the applications are attached as (concurrent) child statemachines of the application statemachine, or descendants thereof. The framework maintains a global *event broker* for event queuing and broadcasting. Event objects can contain various cargo, and are injected in the active statemachines by the event broker component. A *stream broker* component is in charge of the maintenance of the stream graph. It sorts the acyclic Data Flow graph of streammachines, and pushes values through the active connections of the graph.

The discrete domain can influence the continuous by requesting the creation, connection, disabling etc. of streammachines and connections in the stream graph. The continuous domain can influence the discrete domain by broadcasting events to the event broker. This is done through specialized streammachines, e.g. triggers, which generate an event based on the value of incoming streams.

Naturally, functionality in both domains can influence the actor domain through their respective callbacks. This is often necessary, for example to obtain events and streams from the underlying architecture into the StateStream domains, or to reflect states and values through actor properties, e.g. position or visibility status. In section 6.5, we will use several interaction examples to demonstrate these relations.

6.4 StateStream Prototype

In this section, we demonstrate how the StateStream integrates interaction modeling with the components of our VR architecture.

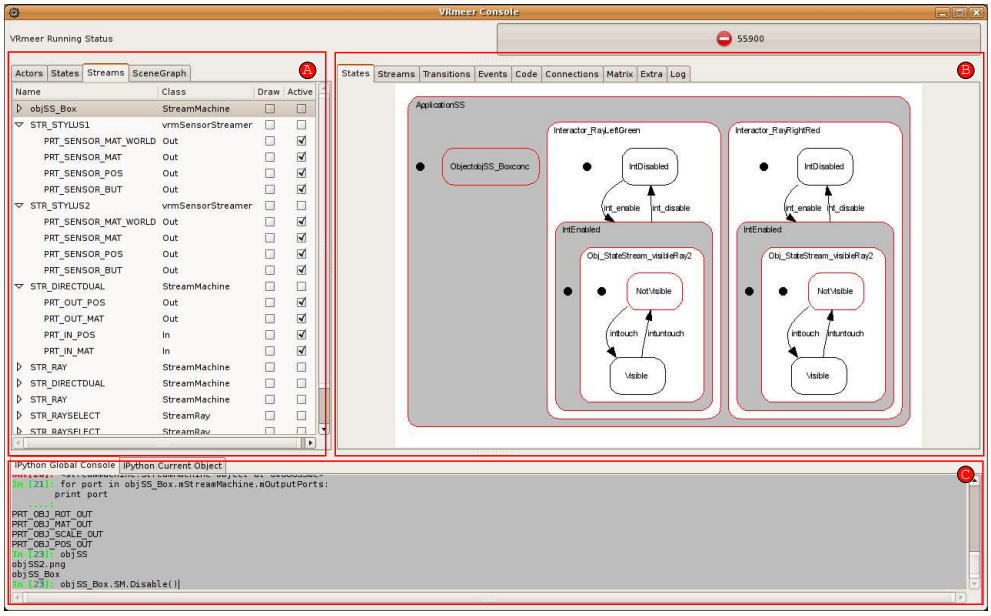


Figure 6.7: Screenshot of a typical layout of the StateStream front-end GUI. Visual widgets such as lists, trees (a) and graphs (b) are grouped in two main panels, and dynamically reflect StateStream state and relations of the current running VR application. New interaction tools and widget elements can be loaded and activated at run-time from the interactive Python shell (c).

6.4.1 Base Architecture

StateStream integrates on a Python level with our in-house VR toolkit *VRMeer*, which mainly builds upon the C++ *OpenSceneGraph* library. We use *SWIG* to create Python bindings for both OpenSceneGraph and the VRMeer toolkit. Through

sets of flexible Python abstraction layers, one can quickly integrate various Python and C++ toolkit functionality in a VR application. A run-time development front-end interface is provided through *iPython* interactive Python shell, optionally integrated with the *PyGTK* GUI toolkit. A more detailed overview of abstraction layers is given in [de Haan 07b], and for the VRMeer architecture with front-end GUI see [de Haan 08a]. Running on top of the base abstraction layers, our proposed StateStream interaction model integrates interaction modeling with other system functionality.

6.4.2 StateStream integration

In our current implementation, a Python interpreter controls the execution of a StateStream enabled VR application. The VR system's main loop interleaves the distribution of the event to statemachines through the event broker, the execution of the streammachine graph through the stream broker, the scene-graph update and the update of an optional GUI. Events can be identified by their name and often accompanied by a cargo. This cargo can denote the destination object(s) and possibly extra parameters. Statemachines and their transitions can use a variety of filters to determine if an event should be processed. For each rendering frame the event broker's queue is processed until it is empty. Cycles in event execution can be detected and avoided.

A set of basic system actors can convert events and variables of underlying scene graph or tracking libraries to abstract, StateStream compatible events and streams. By performing this conversion at a low level, the fine-grained details of interactivity can already be flexibly modeled, composed and inspected through their explicit models. As a result, behavior is modeled *orthogonal* to system abstraction layers, so components in virtually all layers can be used from within control of StateStream primitives. An example of this is the conversion of button presses and pose of a 3D input device, where button presses are converted to events, while positions and rotations are made available through streammachine nodes. Other system actors *monitor* streams to perform, for example, an INSIDE test on a set of VR objects and trigger a TOUCH event as a result.

6.4.3 Front-End Interface

Concurrently with the running VR application, a graphical interface for interactive debugging and development environment is available. It provides an interactive Python shell and visual elements that reflect the internals of the VR application, see Figure 6.7. The availability of the front-end on a separate display or remote computer enables live debugging sessions, which is especially useful for immersive, tracked VR applications. We currently include visual widgets such as lists, trees and graphs that reflect current application and StateStream state and relations. New application-specific widget elements can be created through code and existing layout designers,

and can be loaded and activated at run-time. A powerful approach is to extend widgets to be StateStream actors as well, in order to integrate some of their logic and behavior in the application. For example, a group of text widgets can show the values of an incoming stream connection. We envision that interaction techniques can be made available with GUI panels and a programming API to configure and tune their use.

6.5 Results

In this section, we will give an overview of some of the resulting interaction techniques we have obtained with our system and discuss the process of designing them. We will first illustrate a straightforward VR selection technique to explain the basic working of the system. This description is deliberately kept at a low level, so it will be clear how the flow of execution is. At the same time, this shows that even for a toy interaction technique a textual description can sometimes become quite elaborate, see also [Wingrave 08]. We will then continue and gradually extend them to more elaborate techniques, where combinations of states and streams soon become overwhelming. Also keep in mind, that some simple constructions in StateStream appear more intrusive than direct imperative coding would be for the sake of extension, replacement and reuse in other techniques.

6.5.1 Selection and Manipulation

In direct object selection, a 6DOF stylus is used as a straightforward selection tool of a single VR object. If the tip of the stylus is inside, or touches, the bounding box of a VR object, it should be selected and highlighted. We will gradually work towards a ray casting selection and manipulation technique, so some of the elements described here can be found in the more complex Figure 6.6.

The direct selection actor class is sub-classed from a base *interactor* actor, which represent the display of a cursor icon. Its base statemachine consists of two states ENABLED and DISABLED, and two transitions to switch between them if a INT_ENABLED or INT_DISABLED event is received. In addition, the streammachine provides the connections to position and rotation streams of an interaction device to update the cursor. The ENABLED state of the base statemachine is extended with an INSIDE and NOTINSIDE child state for selection. Two transitions are added: TOUCH to go from NOT_INSIDE to INSIDE, and UNTOUCH to go from INSIDE to NOT_INSIDE.

A simple VR object is also modeled as a StateStream actor. Its statemachine consists of a TOUCHED and NOT_ TOUCHED state, with transitions OBJTOUCH and OBJUNTOUCH, see Figure 6.6. The streammachine contains several ports for both incoming and outgoing streams for position, rotation etc.

If in the direct selection actor an incoming TOUCH event is received from a system actor, the state is changed to INSIDE. In turn, in the state entering function of the IN-

SIDE state, the selected object is sent a OBJTOUCH event. In the NOT_TOUCHED state entering function of the VR object, a highlighting function can be called to change its appearance. Note that in this toy example, we see mainly state-based communication between object and interaction actor.

Ray Casting Extension

To extend the previous example by allowing remote selection with a ray, we have to make a number of changes. First, a ray actor is created that serves to visually represent the indicated direction of the stylus. We create the statemachine of the ray actor in the ENABLED state of the interaction actor, and it contains states to indicate visibility and RAY_HIT or RAY_MISS, see Figure 6.6(b, right sub-state). In the state exit and entry functions of the interactor INSIDE and NOTINSIDE child states, visibility events for the ray are generated.

Second, the ray's streammachine is connected to the interactor's streammachine to communicate starting position and direction. In the value update function of the ray's streammachine, these values are used to update the visual ray object. Third, a system actor is activated that performs the actual ray casting hit algorithm with scene objects. This actor generates TOUCH and NOT_TOUCHED events, but additionally activates a continuous stream for the position of intersection points. The interactor's INSIDE state is extended to create the connection of the intersection point streammachine to an end point port in the ray's streammachine, see Figure 6.6(d). The ray actor itself is responsible for its continuous visual updating. In the RAY_HIT state, we choose to use the end point instead of the direction for drawing. In this example, one observes a clear interplay between discrete, state-based communication but also continuous updating of values through streams.

Object Manipulation

If the stylus button is pressed during selection, one should be able to manipulate it. Once one is manipulating an object, the new object position and rotation are calculated based on the original stylus pose just before the start of the manipulation, and the movement of the stylus. Apart from state changes, object manipulation therefore also requires several continuous communication streams between the object and interaction actors. This is achieved by creating streammachine that, in the TOUCHED state, connects to the stylus pose and to the object pose, and calculate the difference. As soon as the stylus button is clicked to start manipulation, this difference stream is connected to another streammachine that adds it to the new stylus pose stream. The output of this adding streammachine is the new resulting pose of the object. This output is connected to the object's streammachine, and it can use this to update its position.

In this example, one sees a carefully orchestrated combination of discrete and continuous mechanisms. Also we show the use of streammachines as functional

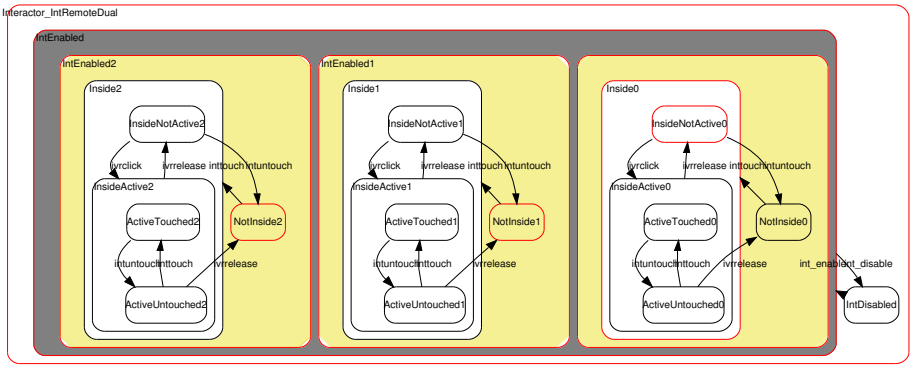


Figure 6.8: A StateChart diagram of a multiple object selection actor. For two recently selected objects and a currently selected object, an instance of a template statemachine was generated. These keep the object-interaction relation consistent.

filters to combine several streams.

6.5.2 Multiple Object Selection

The previous examples work on a single "object to interactor" relation. If the scene contains multiple objects, for example, an interaction technique might be required to manipulate more than one of them simultaneously.

To ensure correct handling, this requires that the interaction actors maintain a separate set of states and streams for each object that it is involved with. To avoid modeling in advance of all possible relations, we can dynamically instantiate a *templated* actor. Each templated child instance reflects a different "object to interactor" relation. As a result, we can extend an existing selection interactor to be used on multiple objects.

To achieve this, events reaching the interactor are intercepted and, if necessary, a new template is instantiated. The template actor creates both a new child statemachine and new streammachines, including all transitions and connections. A templated state of the actor is indicated in Figure 6.8 in yellow. The originally modeled state changes and stream connections still work as advertised without changes. This property is important for extending techniques to work with multiple actors while maintaining overall consistency. In this example, the templated actors have no interaction with other instances. The dynamic instantiation of extra model compontents is made possible by the dynamic features of Python. We are not aware of other systems where similar extension of existing techniques can be done without requiring much manual recoding and bookkeeping.

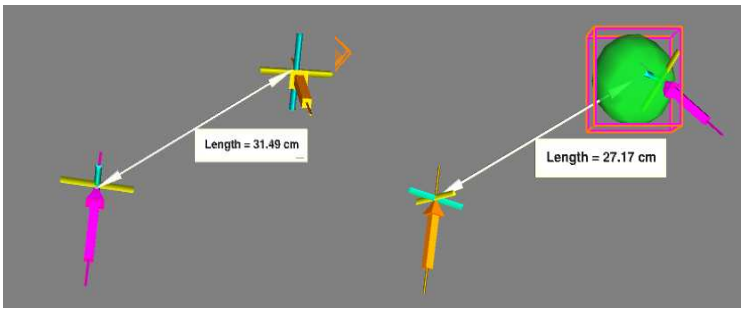


Figure 6.9: Snap Measurement technique. A measurement actor can be controlled by two 3D cursors (left) or the connection of one endpoint snaps to the object by starting a measurement inside the object (right). One cursor concurrently manipulates the object, while the measurement actor remains consistent.

6.5.3 Snap Measurements

This third example demonstrates how an interaction technique can transfer its continuous operation to objects. The interactive element here is a visual measurement tape that can measure distances between arbitrary positions in space or between objects, see Figure 6.9. With one or two styli, a measurement tape can be drawn. In a regular, empty scene, the measurement actor is drawn from stylus tip to the other stylus tip, or from the last clicked point to stylus tip. A panel is positioned and its text updated. When the stylus enters an object however, it interprets the user action as wanting to measure from this object and not in empty space. The stream connection to the stylus interactor is not directly used for drawing anymore, but the object position is used instead. In such a way, a measurement tape actor can be created that measures position between two objects. The advantage here is that the objects can still be moved around while the measurement tape automatically updates. This demonstrates the transparency of the actors, e.g. that the original relation of interactor-to-object can be replaced by object-to-object. This relation is often fixed in other interface implementations. It also gives insight in how streaming can be used to generate constraints dynamically, while the Data Flow graph is kept consistent.

6.5.4 Two-Handed Scaling

The final example combines the earlier examples into a two-handed object scaling technique, similar to the popular two-finger scale technique on multi-touch displays. It extends interaction demonstrated earlier, but uses a tri-fold relation between an object and two interactors. The core modeling technique used here is state ordering. Two interactors are individually used as regular selection and manipulation tools. As described, manipulation updates are communicated through stream connections to the object. When the two manipulate the same object, a “fight” for preference

might occur, as they would normally have parallel state machines and connections as generated from the template. In the object, we want to redirect and combine incoming connections to different properties of the object, e.g. depending on the order of incoming styli. For scaling, the changing distance between the two points of interaction determines the scaling factor of the object, where the first stylus determines the pivot point. Individually, the interaction technique is hand-symmetrical, but it can change to asymmetrical depending on the scaling mode. Through the use of template ordering, the functionality of each object-interactor relation may change. Although this example illustrate the complex interaction between concurrent sub states, for clarity a separate actor to control state ordering might be more comprehensive.

6.5.5 Development Use

We observe that the separation of concerns from the actor, discrete and continuous domain serves as a guide already in an early design phase. Developer discussions tend to focus more on how to get into a certain state, and what connections to ensure in that state. Through containment of difficult situations, overall understanding is enhanced. Also, the development of actor appearance and behavior gets implemented and tested as self-containing components, before they are integrated with other actors. The interactive front-end assists in quick prototyping on top of existing actors, by loading, activating and stepping through situations while observing and adding connections.

A wide array of streammachines is being constructed to provide flexible combinations of streams. In that sense, a bottom-up approach is used to provide a wide platform of tools. Because of the hierarchy in states, also a top-down approach is used. By working from a basic state and gradually specifying new sub states, one can postpone new detailed behavior specification. From use by different developers, several design patterns emerge with a focus on re-use. To improve composition and reuse, it is preferred for actors to maintain as much state as possible, and to consume and replicate streams instead of just connecting external objects. Implementation difficulties mainly arise in conversion of external components. For example, concepts of event systems may differ, such that the absence of an event can be considered an important “idle” event in a different system.

6.6 Discussion

In this section, we will shortly discuss results and limitations of the current model and its integration in the architecture. We are aware that, for a good understanding of the strengths and weaknesses of current approach, we require a long-term evaluation of the development of complex interaction techniques in real-life applications,

on different types of interactive systems. Similar to SwingStates [Appert 06], an evaluation where students implement existing techniques is part of future work. It must be stated that the current model and architecture have already evolved over the last year as a result of iterative application development experience.

First, we currently use basic StateCharts and Data Flow mechanisms to serve as a proof of concept. More elaborate variants of these might be used in the future, such as different StateChart approaches [Beeck 94]. Second, our practical focus is on prototyping: model integration, the avoidance of a pure declarative description and dynamic model changes. Although simple consistency checks can be made on the models, real formal verification and operational semantics are more difficult, especially for dynamically changing models. The model is executed from within a Python interpreter, so off-line verification for full applications will involve complex parsing of Python code trees. Third, some advanced model and introspection properties are currently restricted to dynamic features of the Python interpreter. The use of Groovy[1], a dynamic programming extension for Java, is an interesting alternative. Fourth, good system and model integration results from running a single Python interpreter with a single event broker and stream broker. Performance-wise this is sufficient for most applications and we are investigating optimization and distribution. For systems or stream components that are critical in latency and performance one would have to resort to advanced multi-threading schemes. Finally, because of our developer-centric, language-based approach we currently do not have a unified graphical model representation which can be edited graphically from a full-fledged IDE. Although we consider this essential for a transfer to a larger audience, we see this as a substantial amount of system engineering to be included in a later stage.

6.7 Conclusions and Future Work

In this chapter we described StateStream approach towards unifying the integration of model, architecture and tools for the development of 3D interaction techniques through a dynamic language. The dynamic, interpretative nature of the language provides familiar syntax, and allows us to employ flexible communication between model domains and to introduce novel modeling features, for example the generation and activation of templated behavior descriptions at run-time. We demonstrate the language-based approach on a dualistic model, in an effort to provide two primitives that fit specific discrete and continuous domains of interactive behavior. In this way, we provide structure and separate concerns when designing or studying complex interaction between actors. As demonstrated in several examples, this interplay between the two main model domains and the actor domain is suitable to define complex behavior, but can be extended with the dynamic language. The observation we want to make here is that in practice, aside from toy examples, no single, purely

[1] http://groovy.codehaus.org/

model-based approach can be expected to be complete and elegant. That is, it is inevitable to encounter situations that cannot be modeled within the current model specification, or require a more complex construction pattern. The use of a dynamic programming language and dynamic adaptation of models at run-time can create complex interaction constructions. However, to maintain an overview, custom tools can be integrated to selectively analyze and visualize their hierarchical structure and communication. With this in mind, we feel our iterative integration approach of interaction models, its underlying architecture and development tool sets is a useful asset towards bridging the gap between models and practical use.

The StateStream system, model, syntax and tools are continuously refined. Naturally, we strive to enlarge the set of available actors and front-end tool widgets to enhance developer productivity. From a research perspective, we aim at more understanding of the interaction model to allow reasoning on actor relations. We believe that this understanding is needed for extending the current state-of-the-art with optimization and distribution over multiple processes or machines. Finally, we plan to investigate alternative visualization techniques to enable better insight and evaluation of interaction techniques over time.

7 Conclusions and Discussions

In this thesis we described work in the area of 3D interaction. We focused on two themes: designing techniques with the VR system characteristics in mind (Part I), and iterative development of applications and interaction (Part II). In this final chapter, we summarize the main results and contributions, discuss the current state of the work and give our view on future directions of research.

7.1 Research Questions and Contributions

In the introduction chapter we have indicated the role of 3D user interaction in the context of interactive 3D visualization. The work presented in this thesis has shown that real VR (visualization) applications and their 3D user interaction techniques need careful attention. This is apparent in both the creation process of the software and in the final usability of the application and its interaction tools. We formulated two main research questions. In this section we reiterate the main contributions that were presented to address these research questions. In the following section 7.2.1, we more explicitly reflect on these contributions and how they are embedded in the current state of the work.

Part I of this thesis concerns the design of 3D interaction techniques. Following the first research question, we aimed to develop 3D interaction techniques that are "stable" with respect to inaccuracies in human manual capabilities and in display and tracking devices. The following contributions were presented to address the

research question:

- IntenSelect, a customizable 3D object selection technique (Chapter 2). This technique extends selection-by-volume approaches and uses a flexible ray to provide selection feedback. The main novelty of this technique is the ability to configure the selection scoring metric and the use of time-dependent object ranking. In this way, it does not only introduce more stable spatial control of selections, but also serves as a stabilization for input and object movements over time.

- Generic hybrid 2D/3D interface concepts for medium sized VR platforms (Chapter 3). Hybrid interfaces offer familiar 2D UI windows and controls in a 3D environment. The main contribution lies in the extension of IntenSelect and its application for 3D selection and manipulation of these controls. These extensions include direct selection, object snapping feedback and spring-based manipulation. When used in the 3D environment, the 2D interface elements are not only visually integrated, but also the interaction with 3D input devices on them is seamless.

- Analysis and solutions for view sharing in multi-user VEs with special attention to co-location, object selection and manipulation (Chapter 4). This contribution extends earlier work on multi-user interaction on projection based VR. Possible solutions for single-view camera models and compensation techniques are proposed and implemented. In the context of basic, close-range object selection and manipulation tasks, effects of view distortion on task performance and accuracy are demonstrated. Results indicate that our alternative camera models can provide a more stable overall collaborative user experience.

It is acknowledged that spatial interaction within a Virtual Environment has its limitations, but can be overcome to some extent. In summary, one should not expect or require high physical precise input from users, and, at the same time one does not need to offer a perfect visual result. Users are capable of and used to dealing with "fuzziness", and as such a 3D interface can benefit from this.

Part II of this thesis concerns the development of 3D interaction techniques and the supporting software architecture and models. Following from the second research question, we aimed to support generic prototyping for fast development of and testing of interactive 3D visualization applications and their interaction techniques in a Virtual Environment. For application and interaction development we introduced an architecture to improve software flexibility. Again, the goal here is to facilitate the fast, step-by-step, creation and evaluation of an application. The following contributions were presented to address these issues:

- A software approach to promote a flexible development style for VR visualization applications (Chapter 5). The use of a single abstraction language allows

integration of heterogeneous components and external software at different layers in existing VR frameworks. The main contribution here is the interactive scripting environment that gives run-time access to the running VR applications, providing an interactive prototyping Virtual Environment.

- StateStream, a developer-centric approach to facilitate iterative design and development of complex interaction techniques and scenarios (Chapter 6). The interaction mechanisms are described in structured code, using classes to model primitives of both states and streams. The main contribution is the tight integration of this interaction description model, the run-time architecture and development tools through a dynamic language. The power of this integrated approach is demonstrated by the possibility to visually observe and interactively adapt the internal behavior of an interaction technique while it is being used in an application.

In the following sections we will discuss the current status of the work presented.

7.2 Current State of the Work

In this section we will discuss the current state of the work. It discusses developments that followed initial publications of the articles presented in the chapters, and indicates the relations between the individual chapters.

7.2.1 Interaction techniques

The presented IntenSelect technique in Chapter 2 is currently integrated in two of our main scientific data exploration applications we described in the first Chapter, namely MolDRIVE and CloudExplorer. Informal observations from this integration in practice reflect the impression we got from the user experiments: users appear faster and more confident in making selections. In the CloudExplorer application it is of value because it allows the use of many small hybrid interface elements for operating the windows, buttons and sliders, see Chapter 3. For use in the MolDRIVE application IntenSelect was extended to handle the selection of the many moving individual atoms separately. Although it is considered very helpful in this application, difficultly remains when the depth-complexity is very high. This occurs in situations such as exploring large, dense molecules, where ten or more atoms frequently line up along the selection ray. One way to cope with selections of these types is to extend the scoring mechanisms by grouping into higher-order primitives. In the case of MolDRIVE, one could group atoms into the *amino acids* they belong to, and use an overall score for the group.

The ideas behind IntenSelect were also picked up by several other researchers. Our extensive discussions with Anthony Steed on the design of IntenSelect and possible extensions have contributed to a more general, abstract model for selection.

Steed proposes an abstract model for selection that is based on time-varying scalar fields (TVSFs), with the purpose of highlighting potential areas for development and evaluation of novel selection techniques [Steed 06]. In other work, Hentschel et al. use IntenSelect to select particles in the blood flow simulation of a Ventrical Assist Devices [Hentschel 08].

The concepts for multi-user viewing and interaction, as presented in Chapter 4, are currently used within the MolDRIVE application. During experiments and the many demonstrations, we observe that this immediately invites users to concurrently interact with the molecules and the interface elements. In terms of developments on hardware-based multi-user displays [Fröhlich 05], the concepts of shared views and interaction can also be useful. For example, when a two-way independent stereoscopic display (e.g. a round tabletop) is used with more than two users, one might dynamically group those users that are close together to share a single view. An intriguing element in the MolDRIVE application is that IntenSelect is enabled together with the multi-user viewing and interaction. Although we have not performed user experiments on this combination, we believe that IntenSelect does ease the selection of objects from moving, distorted viewpoints. This integration process of the two techniques was performed manually in the original C++ code base, was error prone and currently difficult to extend. This issue was the main motivation for the development of the StateStream approach. In the near future, we plan to reimplement the integration using StateStream and carefully inspect its development cycle.

In Part I we have conducted two user experiments. The main reason to perform these experiments is to make initial, formative judgments for design guidance of the techniques and their parameters. This is in contrast to a more formal, evaluative judgments that aim for explicit quantitative comparison with sufficient statistical power. An apparent drawback from our approach is that we cannot make strong, definite statements on the quality of the proposed techniques. However, the results from our small quantitative experiments complement qualitative interviews and observations of use in illustrative example applications. Combined, all of these assist in clarifying user performance in VR applications in which the techniques are embedded.

There are many configuration parameters and external factors that influence the performance of the interaction techniques. To achieve a more accurate evaluation in future, these parameters and their interplay should first be studied separately. For example, the IntenSelect technique would require a more in-depth analysis on the influence of parameters such as stickyness, as well as on external factors such as user distance, latency of tracking, screen size and foremost the scene complexity. The added value of future experiments, and the possibility for generalization of results to real applications and other VR systems, depends on how well these individual influences can be parameterized in the experimental design. In other words, the

techniques and the testing environment first have to be more stable and mature to obtain more general evaluative power from user experiments. We expect that the findings from a more in-depth analysis can also assist in constructing a procedure for parameter calibration. This procedure could be used to find a good configuration of parameters given a certain application on a specific VR system.

7.2.2 Architectures for Interaction

The development architecture with flexible abstraction layers, discussed in Chapter 5 has enabled us to integrate and experiment with VR applications more easily. The wrapping effort has had considerable leverage, as it allows more accessible and integrated use of existing work. We must state that the wrapping process itself remains tedious and requires maintenance. This is specifically a practical software engineering problem, and its technical details and workflow deserve more careful attention.

The mature stage of software is not quite reached, so no large-scale, statistical user testing or comparison with alternative approaches were performed. A recent, similar approach to our design, the HECTOR system [Wetzstein 07], appeared in parallel to our work but is no longer under active development. For reference, we do maintain a list of toolkits with scripting capabilities on a web page[1]. For our approach, we made informal observations of its use throughout the project to guide its development. The Python-based development approach was used in the evaluation of the multi-user techniques, as described in Chapter 4. Here, the integration with Python of simple user logging and external software toolkits such as *R* and *matplotlib* streamlined the generation and analysis of experimental results. So far, we had three computer science students work with the system for a longer period of time. The student Berend Wouda implemented the well-known GoGo selection technique within a small assignment for a computer graphics course. Two students worked with it during their Master's graduation work. René Molenaar worked on the evaluation of the multi-user techniques using the Python layers. Based on our work on user interfaces for video surveillance [de Haan 09b], the student Huib Piguillet reimplemented the original navigation techniques in Python, which can now more easily be extended and adapted. Finally, we have used the system in four one-day sessions for PhD students as a part of a post-graduate course on data visualization and VR. The accessible syntax and semantics take away much of the original learning barriers of the C++-based scene graph. We observed for example, that it is possible for novice users to explore and add functionality in VR data visualization applications in a matter of hours.

The main target for the wrapping strategy and the abstraction layers was to enable flexibility in the development of interaction tools. Because the exact software

[1] http://visualisation.tudelft.nl/VRdev

requirements for interaction models are often not clear, StateStream itself was iteratively designed and developed on the Python-based abstraction layers. An interesting observation is that during the described transition to OpenSceneGraph, StateStream-based tools were relatively easy to convert. In most cases, only scene graph-specific components of streaming modules and the interaction techniques needed to be adapted.

We are currently in the process of redesigning and combining existing interaction tools with our StateStream approach. In a new application for exploration of 3D topographic data such as detailed terrain maps and point clouds, we are using StateStream to develop all interaction techniques, see also section 7.3. As we go, new streaming components are being developed, and existing interaction techniques are recreated. From early results, we see that the StateStream approach allows us to specify interaction behavior at low level, and invites one to experiment with slight variations and combinations of existing techniques. To study the creation process of 3D interaction techniques in more detail, it will be worthwhile to let various pairs of developers and designers reimplement or adapt some existing, popular interaction techniques using StateStream. After this one can study and evaluate both the artifacts of the process, such as sketches, storyboards, discussions and models, as well as observe emerging design and development patterns. From this evaluation, the necessary software tools and model primitives can be adapted. Through this approach, we hope to gain more insight in the various factors that affect user's experience and performance, and ultimately design more effective exploration tools.

7.3 New 3D Devices and Applications

Within the context of this thesis several hardware devices and application case studies were addressed. In this section the major cases are briefly discussed, and they are illustrated in Figure 7.2.

We designed and built the *PDRIVE*, a Projector-based, Desktop, Reach-In Virtual Environment [de Haan 07a]. In this *active space* VR system, a user "reaches in" to the virtual environment, which he views through a mirror, see Figure 7.1 (left). Two standard, off-the-shelf DLP projectors generate stereo images and provide a large display and workspace volume. The main novelty of this system is the compact configuration of off-the-shelf office projectors and mirrors, making suitable for on-desktop at relatively low cost. At the time of design, most similar systems used CRT monitors and were limited to a certain screen diameter and LCD panels with stereo capabilities were not produced yet. Three prototypes were built with different screen or projector characteristics. Currently, a more attractive alternative would be to instead use a stereo-enabled LCD panel for display. Another attractive alternative would be to create a two-user PDRIVE by replacing the current projectors with stereo-enabled ones.

The use of the *Wii Balance Board* as a low-cost VR input device [de Haan 08b] was

Figure 7.1: Two new devices for 3D interaction: The PDRIVE VR system we designed and built (left) and the *Wii balance board*, a no-hands input device controlled by feet pressure (right).

explored. In many interaction scenarios, the user's hands are already occupied with a primary task, such as object selection. Transferring secondary tasks, such as travel, to the feet avoids mode-switching and possibly improves the application's usability. The Wii Balance Board, see Figure 7.1 (right), measures pressure values from under the feet of the user, and is targeted as an input device for the Wii gaming console. By processing the pressure sensor values from the balance board on a PC, we were able to use it for both discrete and continuous input in a variety of VR interaction metaphors. Controlling first-person travel by standing on the board comes to mind, but we also explored other scenarios and interaction techniques, including using the balance board while seated. The discrete input is suitable for control input, such as mode switching or object selection. Using continuous input, the balance board is well suited for interactions requiring two simultaneous degrees of freedom and up to three total degrees of freedom, such as navigation or rotation. The initial publication and software release triggered responses of several parties with interest in the use of the software. Their envisioned target applications include navigation in Virtual Environments for soldiers, balance training games for elderly and research into the effects of drugs on stability and the center-of-gravity.

The potential benefits of Virtual Environments for use in video surveillance tasks were explored [de Haan 09b]. Current surveillance systems can display many individual video streams within spatial context in a 2D map or 3D Virtual Environment (VE). The aim of this is to overcome some problems in traditional systems, e.g. to avoid intensive mental effort to maintain orientation and to ease tracking of motions between different screens. However, such integrated environments introduce new challenges in navigation and comprehensive viewing, caused by imperfect video alignment and complex 3D interaction. A novel, first-person viewing and navigation interface for integrated surveillance monitoring in a VE was proposed. It is cur-

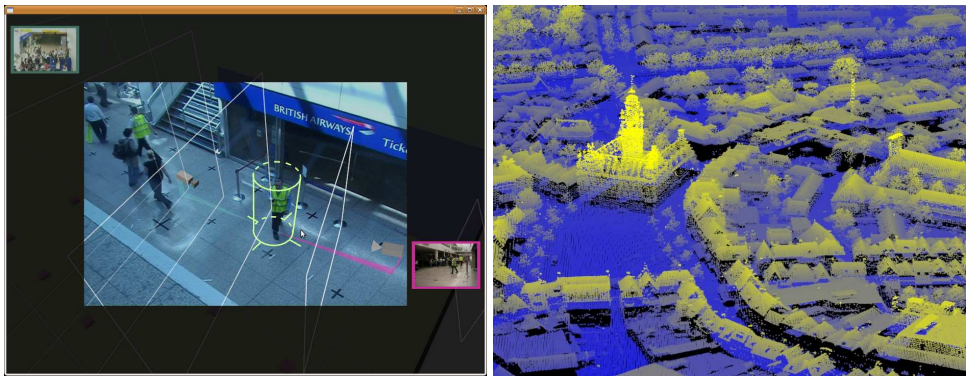


Figure 7.2: Two new application prototypes that employ 3D interaction: navigating surveillance videos in a VE (left), and large 3D pointcloud exploration (right).

rently designed for egocentric tasks, such as tracking persons or vehicles along several cameras. For these tasks, it aims to minimize the operator's 3D navigation effort while maximizing coherence between video streams and spatial context. The user can easily navigate between adjacent camera views and is guided along 3D guidance paths. To achieve visual coherence, we use *dynamic video embedding*: according to the viewer's position, translucent 3D video canvases are smoothly transformed and blended in the simplified 3D environment. The animated first-person view provides fluent visual flow which facilitates easier maintenance of orientation and can aid in spatial awareness. It is interesting to note that there is no perfect 3D reconstruction of the virtual world from the camera views. Instead, subtle visual effects (motion, blending) are used to "hide" this fact during camera transitions. Nevertheless, these provide a 3D sensation that appears to be sufficient for spatial understanding and interaction. As a follow-up to this application, a second prototype was rebuilt by Msc student Huib Piguillet using our Python layers on OpenSceneGraph. In this prototype the use of various augmented 3D overlays and *through-the-lens* navigation controls are explored, see Figure 7.2 (left).

In the context of a *3D topography project*, the interactive 3D visualization of large 3D topographic models, such as terrains and point cloud data, was explored. A StateStream-based application was used to design the interactivity for navigation and annotation. For navigation, different input devices such as a mouse, a tablet, the *SpaceMouse*, and the balance board were used in combination with stereoscopic rendering on the PLANAR display. In addition, investigations concentrated on the datastructures for streaming and rendering of the large terrain data and 3D point clouds. The target data set consists of height maps and laser-scanned point-clouds of a part of the Netherlands. We preprocess the data in order to create Level-of-Detail structures that can be streamed over the network. In our most recent proto-

type application, we can explore both the large triangulated terrains and point cloud datasets at interactive rates in 3D, see Figure 7.2 (right). The exact scalability and performance of these techniques are under current investigation.

7.4 Software Development and Maintenance

As a necessary part of this thesis work, a large investment was put into in the development and maintenance of our software toolkits. The development started based on the commercial OpenGL Performer toolkit and our in-house RWB library. Early in the project, the RWB library was refactored and renamed to IVR toolkit, to facilitate better interaction support. Both the interaction techniques and flexible abstraction layers were initially implemented on top of Performer and the IVR toolkit. The commercial Performer package limited the interchange and distribution of code, and was unfortunately abandoned by its manufacturer SGI. Two years into the course of this thesis project, most functionality and applications were transferred to the OpenSceneGraph toolkit. This transition has led to the final VRMeer codebase, and contributed to the *osgSWIG*[2] project for wrapping OpenSceneGraph. Recently, the largest part of the VRMeer was released[3] under the BSD open-source source license. Currently we are able to distribute and share large parts of our code to research partners, and we co-maintain the osgSWIG project. The transition to OpenSceneGraph implied considerable extra development effort. However, the transition process allowed us to make new design decisions and to confirm that the layered approach works on a different scene graph toolkit as well.

7.5 Future Work

An ongoing part of the work is the optimization of new and existing interfaces and 3D interaction techniques. This includes customization for a specific application, the set of interaction tasks involved and the display and input devices in use. At the same time, this includes personalization and adaptation to the level of experience. For instance, we expect that the parameters for IntenSelect can be gradually adapted to facilitate experienced and skilled users with faster movements. During this process, it is advisable to regularly involve experts in other areas, such as interaction designers, application domain-experts and potential end-users. We expect that the improved flexibility and visibility of the programming layers and the interaction model will contribute to a more effective communication between all parties involved.

[2] http://code.google.com/p/osgswig
[3] http://visualization.tudelft.nl/VRmeer

For a greater applicability and integration of the external software, a collection of pre-built sets of components and interaction techniques could be provided. An example of this is a set of Python-based and StateStream components for using a video source such as a webcam as an input device. Next to the code, also graphical representations such as the state and stream graphs can be provided to serve as additional documentation of the interaction techniques. To increase user acceptance at a higher level, a graphical 3D interaction builder could be conceived.

Although a high update rate and low latency is necessary for 3D interaction techniques in VR, our described approaches do not focus on performance. We anticipate that through analyzing the explicitly modeled characteristics of an interaction technique, performance optimization can actually be addressed more effectively. First, one can perform an analysis of a streaming network of an interaction technique in order to optimize the frequency and order of execution, or to minimize latency from input device to visible result on the screen. For example, a line-drawing component can be provided with smooth data from a tracking device running at 200 frames per second, while its visual component for rendering update runs at 60 frames per seconds. Second, one can expand some states and their associated stream networks of an interaction technique, in order to extract regular sequential code that can be used in non-StateStream applications. In this way, interaction techniques or parts of them can be prototyped within StateStream and later exported and converted manually into high-performance C++ code. Third, of special interest is the extension of Python and StateStream based mechanisms for multi-threading and distributed, networked systems. An example here can be a multi-system distributed VR application, where the IntenSelect technique accommodates corrections for latency in the scoring functions.

We have designed our 3D interaction tools and models within the main context of VR/Vis applications. A next step in the development is to extend tools and behavioral models beyond interaction. We can consider the entire real-time, interactive visualization loop from 3D interaction, to visualization algorithm and the various fast rendering methods. Visualization algorithms and rendering techniques can make their states and properties explicitly available for integration with interaction mechanisms such as StateStream. In this way, a visualization algorithm could be redesigned with direct 3D interaction in mind. For example, an iso-surface generator could have a state in which it has fast, interactive drawing from the 3D position of a stylus, and a slow, high-quality state. Also here the analysis of connections can be used to address performance optimization. For instance, one can introduce techniques for progressive quality rendering of visualization elements, latency layering, and rendering on a time budget. An interesting approach could be the integration of techniques for multi-framerate rendering [Smit 08].

7.6 Vision on Interactive 3D Exploration

In this thesis we have considered 3D Interaction in the context of (semi-) immersive Virtual Reality systems. Also in [Bowman 08], it is stated that 3D UI research has been tightly connected to VR. We indeed acknowledge the trend that many of the lessons learned on interaction in fully equipped VR systems can also be applied to non-immersive or desktop systems. For instance, we also use 3D UIs in desktop applications, see [de Haan 09b], although these are controlled by a regular mouse and viewed on a regular display. On the other hand we see that, under pressure of the entertainment market, new impulses of novel 3D displays and multi-user, spatial input hardware bring the characteristic potential elements of VR within reach of many. We feel 3D interactive systems will not replace standards in 2D desktop environments, but will be used as an addition when 3D data needs to be explored. To achieve this, we should strive to embrace new possibilities by integrating them in the regular workflow of the end-user. In this light, we feel the following issues to be of interest to future research challenges:

- *Live Simulations and Sensors.* Developments in computing methods and networks continuously bring new types of 3D data, previously only to be explored off-line, within reach for interactive live exploration. With the introduction of new generations of parallel processors (e.g. Cell, GPUs), one can interactively perform many physical 3D simulations at a meaningful resolution and timescale. When combined with interactive visualization we can directly explore and steer these. Just as with the simulations, live sensor network integration that feed information in virtual worlds become feasible. For example, (3D) video streams from many networked cameras can be integrated and explored. In both of these cases this will introduce a shift in workflow from off-line, batch-oriented processing to live and interactive analysis. Especially here, intuitive 3D exploration tools are key to fast insight and decision-making. The challenge is how to represent live information and how to interactively steer simulations based on new sensor input.

- *Get together and interact.* The trend in multi-user interaction was set in the last few years with the introduction of affordable multi-touch displays, initiated by Han [Han 05]. It is important to note that the powerful collaborative aspect of discussion and exploration of these systems have put them on the map with the general public. However, interaction issues similar to those in collaborative VR, as well as software architectures to address these, are emerging, see for example [Echtler 08]. We expect that this renewed interest will transfer to multi-user and multi-display solutions for 3D exploration. Also for these new collaborative 3D systems, we expect our StateStream approach to be helpful in addressing the challenge of enhanced viewing and interaction mechanisms.

- *Commuting between exploration and analysis.* The previous two challenges already hinted on the value of exploration and collaboration in VR. To integrate exploration sessions in a more traditional, analysis-oriented workflow, facilities should be available to assist transitions from and to sessions on the desktop environment. Ideally, a desktop-based version of the application is available with a standard 2D interface to prepare data, interaction tools and environment for the exploration session. For example, the setting up of interesting visualization parameters can be done by using *lookmarks*[Stanton 04]. The results of the actual (multi-user, live) VR exploration process can be recorded, including storing tracking of viewpoint and interaction, but also explicit annotations and speech or video. This data can be considered important meta-data, in a similar fashion to using *provenance* in setting up a visualization pipeline [Scheidegger 08]. These exploration sessions can be replayed, shared and later analyzed off-line in detail on the desktop or on the VR system. The challenge here is to provide usable facilities for managing sessions of exploration, and distilling insights from them afterwards.

In this thesis 3D interaction was addressed in two themes: 3D interaction techniques from the explorer's usability perspective, and their development from the designer's or developer's perspective. The possibilities in tools and methods provided to the designer and developer allow them to design and evaluate new, complex possibilities for the explorer's tools. We strongly feel that these two themes are tightly interconnected and will eventually converge. In addition, we feel the presented solutions are not restricted to 3D interaction but extend to other highly interactive systems. We envision that general "interaction engineering" concepts will emerge, designed by new a breed of interaction engineers with interdisciplinary skills who address issues of highly interactive systems. The interaction engineering environments of the future will combine a wide spectrum of (visual) interaction abstractions, and will bring effective (3D) interaction design, development and evaluation closer to the end-user.

Bibliography

[Agrawala 97] Maneesh Agrawala, Andrew C. Beers, Ian McDowall, Bernd Fröhlich, Mark Bolas & Pat Hanrahan. *The two-user Responsive Workbench: support for collaboration through individual views of a shared space*. In Proc. ACM SIGGRAPH, pages 327–332, 1997.

[Allard 05] J. Allard, C. Ménier, E. Boyer & B. Raffin. *Running Large VR Applications on a PC Cluster: the FlowVR Experience*. In Proc. of EGVE/IPT '05, pages 59–68, 2005.

[Angus 95a] Ian G. Angus & Henry A. Sowizral. *Embedding the 2D Interaction Metaphor in a Real 3D Virtual Environment*. In Proceedings of Stereoscopic displays and virtual reality systems II, pages 282–293, 1995.

[Angus 95b] Ian G. Angus & Henry A. Sowizral. *VRMosaic: WEB Access from within a Virtual Environment*. In Proceedings of the 1995 IEEE Symposium on Information Visualization, page 59, 1995.

[Appert 06] Caroline Appert & Michel Beaudouin-Lafon. *SwingStates: adding state machines to the swing toolkit*. In Proc. of UIST '06, pages 319–322, 2006.

[Arthur 98] Kevin Arthur, Timothy Preston, Russell Taylor, Frederick Brooks, Mary Whitton & William Wright. *Designing and Building the PIT: a Head-Tracked Stereo Workspace for Two Users*. Technical report, Chapel Hill, NC, USA, 1998.

[Backman 05] A. Backman. *Colosseum3D: Authoring Framework for Virtual Environments*. In Erik Kjems & Roland Blach, editors, Proceedings of the 9th IPT and 11th Eurographics VE Workshop (EGVE) '05, 2005.

[Beazley 03] D. M. Beazley. *Automated scientific software scripting with SWIG*. Future Gener. Comput. Syst., vol. 19, no. 5, pages 599–609, 2003.

[Beeck 94] Michael von der Beeck. *A Comparison of Statecharts Variants*. In Proc. of ProCoS '94, pages 128–148, 1994.

[Bierbaum 01] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker & C. Cruz-Neira. *VR Juggler: a virtual platform for virtual reality applicationdevelopment*. In Proc. Virtual Reality '01, pages 89–96, Yokohama, Japan, 2001.

[Bimber 01] Oliver Bimber, Bernd Fröhlich, Dieter Schmalstieg & L. Miguel Encarnacao. *The Virtual Showcase*. IEEE Computer Graphics and Applications, vol. 21, no. 6, pages 48–55, 2001.

[Blach 98] Roland Blach, Jürgen Landauer, Angela Rösch & Andreas Simon. *A highly flexible virtual reality system*. Future Gener. Comput. Syst., vol. 14, no. 3-4, pages 167–178, 1998.

[Blanch 06] Renaud Blanch & Michel Beaudouin-Lafon. *Programming rich interactions using the hierarchical state machine toolkit*. In Proc. AVI '06, pages 51–58, 2006.

[Blom 02] Kristopher Blom, Gary Lindahl & Carolina Cruz-Neira. *Multiple Active Viewers in Projection-Based Immersive Environments*. In Proceedings, Seventh Annual Symposium on Immersive Projection Technology (IPT 2002), 2002.

[Bolas 04] Mark Bolas, Ian McDowall & Dan Corr. *New Research and Explorations into Multiuser Immersive Display Systems*. IEEE Comput. Graph. Appl., vol. 24, no. 1, pages 18–21, 2004.

[Botha 04] Charl P. Botha. *DeVIDE: The Delft Visualisation and Image processing Development Environment*. Technical report, Delft Technical University, 2004.

[Bowman 98] Doug A. Bowman, Larry F. Hodges & Jay Bolter. *The Virtual Venue: User-Computer Interaction in Information-Rich Virtual Environments*. Presence: Teleoperators & Virtual Environments, vol. 7, no. 5, pages 478–493, 1998.

[Bowman 99] Doug A. Bowman, Jean Wineman & Larry F. Hodges. *The Educational Value of an Information-Rich Virtual Environment*. Presence: Teleoperators & Virtual Environments, vol. 8, no. 3, pages 316–331, 1999.

[Bowman 01] D. Bowman, C. Wingrave, J. Campbell & V. Ly. *Using Pinch Gloves for both Natural and Abstract Interaction Techniques in Virtual Environments*. In Proc. HCI International, pages 629–633, 2001.

[Bowman 04] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola & Ivan Poupyrev. 3D User Interfaces: Theory and Practice. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[Bowman 08] Doug A. Bowman, Sabine Coquillart, Bernd Fröhlich, Michitaka Hirose, Yoshifumi Kitamura, Kiyoshi Kiyokawa & Wolfgang Stuerzlinger. *3D User Interfaces: New Directions and Perspectives*. IEEE Comput. Graph. Appl., vol. 28, no. 6, pages 20–36, 2008.

[Bryson 96] Steve Bryson. *Virtual reality in scientific visualization*. Commun. ACM, vol. 39, no. 5, pages 62–71, 1996.

[Burdea 03] Grigore C. Burdea & Philippe Coiffet. Virtual Reality Technology. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[Burrows 05] Tony Burrows & David England. *YABLE - yet another behaviour language*. In Web3D '05: Proceedings of the tenth international conference on 3D Web technology, pages 65–73, New York, NY, USA, 2005. ACM Press.

[Carey 97] Rikk Carey & Gavin Bell. The annotated VRML 2.0 reference manual. Addison-Wesley Longman Ltd., Essex, UK, UK, 1997.

[Carr 94] David A. Carr. *Specification of interface interaction objects*. In Proc. of CHI '94, pages 372–378, 1994.

[Coninx 97] K. Coninx, F. Van Reeth & E. Flerackers. *A Hybrid 2D / 3D User Interface for Immersive Object Modeling*. In Proceedings of the 1997 Conference on Computer Graphics International, page 47, 1997.

[Conway 97] Matthew J. Conway & Randy Pausch. *Alice: easy to learn interactive 3D graphics*. SIGGRAPH Comput. Graph., vol. 31, no. 3, pages 58–59, 1997.

[Cremer 95] James Cremer, Joseph Kearney & Yiannis Papelis. *HCSM: a framework for behavior and scenario control in virtual environments*. ACM Trans. Model. Comput. Simul., vol. 5, no. 3, pages 242–267, 1995.

[Cuppens 04] Erwin Cuppens, Chris Raymaekers & Karin Coninx. *VRIXML: A User Interface Description Language for Virtual Environments*. In Proceedings of Developing User Interfaces with XML: Advances on User Interface Description, pages 111–117, 2004.

[Dang 05] Nguyen-Thong Dang. *The Selection-By-Volume Approach: Using Geometric Shape and 2D Menu System for 3D Object Selection*. In Doug Bowman, Bernd Fröhlich, Yoshifumi Kitamura & Wolfgang Sturzlinger, editors, Proc. of the IEEE VR Workshop on New Directions in 3D User Interfaces, pages 65–68, 2005.

[Dix 08] Alan Dix, Jair Leite & Adrian Friday. *XSED — XML-Based Description of Status—Event Components and Systems*. In Proc. of EIS 2007, pages 210–226, 2008.

[Dykstra 94] Phillip Dykstra. *X11 in Virtual Environments: Combining Computer Interaction Methodologies*. The X-Resource, vol. 9, no. 1, pages 195–204, 1994.

[Echtler 08] Florian Echtler & Gudrun Klinker. *A multitouch software architecture*. In NordiCHI '08: Proceedings of the 5th Nordic conference on Human-computer interaction, pages 463–466, New York, NY, USA, 2008. ACM.

[Ellson 03] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North & G. Woodhull. *Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools*. In M. Junger & P. Mutzel, editors, Graph Drawing Software, pages 127–148. Springer-Verlag, 2003.

[Feiner 93] Steven Feiner, Blair MacIntyre, Marcus Haupt & Eliot Solomon. *Windows on the World: 2D Windows for 3D Augmented Reality*. In Proceedings of the 6th annual ACM symposium on User interface software and technology, pages 145–155, 1993.

[Fels 97] Sidney Fels, Silvio Esser, Armin Bruderlin & Kenji Mase. *InvenTcl: making Open Inventor interpretive with Tcl/[incr Tcl]*. In SIGGRAPH '97: ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97, page 191, New York, NY, USA, 1997. ACM.

[Figueroa 02] Pablo Figueroa, Mark Green & H. James Hoover. *InTml: a description language for VR applications*. In Proc. of Web3D '02, pages 53–58, 2002.

[Finkenzeller 03] D. Finkenzeller, M. Baas, S. Thüring, S. Yigit & A. Schmitt. *VISUM: A VR System for the Interactive and Dynamics Simulation of Mechatronics Systems*. In Proc. Virtual Concept 2003, Nov 2003.

[Fisher 86] S. S. Fisher, M. McGreevy, J. Humphries & W. Robinett. *Virtual Environment Display System*. In Proceedings of the 1986 workshop on Interactive 3D graphics, pages 77–87, 1986.

[Forsberg 96] A. Forsberg, K. Herndon & Zeleznik R. . *Aperture based selection for immersive virtual environments*. In Proc. of the Symposium on User Interface Software and Technology, pages 95–96, 1996.

[Frees 05] S. Frees & G. D. Kessler. *Precise and Rapid Interaction through Scaled Manipulation in Immersive Virtual Environments*. In Proc. of IEEE Virtual Reality Conference, pages 99–106, 2005.

[Fröhlich 05] Bernd Fröhlich, Jan Hochstrate, Jörg Hoffmann, Karsten Klüger, Roland Blach, Matthias Bues & Oliver Stefani. *Implementing multi-viewer stereo displays*. In Proc. WSCG '05, pages 139–146, 2005.

[Goslin 04] Mike Goslin & Mark R. Mine. *The Panda3D Graphics Engine*. Computer, vol. 37, no. 10, pages 112–114, 2004.

[Griffith 05] Eric J. Griffith, Frits H. Post, Michal Koutek, Thijs Heus & Harm J. J. Jonker. *Feature Tracking in VR for Cumulus Cloud Life-Cycle Studies*. In Erik Kjems & Roland Blach, editors, Proceedings of the 9th IPT and 11th Eurographics VE Workshop (EGVE) '05, pages 121–128, October 2005.

[de Haan 02] Gerwin de Haan, Michal Koutek & Frits H. Post. *Towards Intuitive Exploration Tools for Data Visualization in VR*. In Proceedings of the ACM symposium on Virtual reality software and technology, pages 105–112, 2002.

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

[de Haan 06] Gerwin de Haan, Eric J. Griffith, Michal Koutek & Frits H. Post. *Hybrid Interfaces in VEs: Intent and Interaction*. In Roger Hubbold & Ming Lin, editors, Virtual Environments 2006, pages 109–118, 2006.

[de Haan 07a] G. de Haan, E.J. Griffith, M. Koutek & F.H. Post. *PDRIVE: The Projector-based, Desktop, Reach-In Virtual Environment*. In B. Fröhlich, R. Blach, & R. van Liere, editors, Proceedings of the 10th IPT and 13th Eurographics VE Workshop (EGVE) '07, 2007.

[de Haan 07b] Gerwin de Haan, Michal Koutek & Frits H. Post. *Flexible Abstraction Layers for VR application development*. In Proceedings of IEEE Virtual Reality 2007, pages 239–242, 2007.

[de Haan 07c] Gerwin de Haan, René Molenaar, Michal Koutek & Frits H. Post. *Consistent Viewing and Interaction for Multiple Users in Projection-Based VR Systems*. Computer Graphics Forum, vol. 26, no. 3, pages 695–704, 2007.

[de Haan 08a] G. de Haan & F. H. Post. *Flexible Architecture for the Development of Realtime Interaction Behavior*. In Proc. of SEARIS '08, pages 71–75, 2008.

[de Haan 08b] Gerwin de Haan, Eric J. Griffith & Frits H. Post. *Using the Wii Balance Board^{TM}as a low-cost VR interaction device*. In VRST '08: Proceedings of the 2008 ACM symposium on Virtual reality software and technology, pages 289–290, New York, NY, USA, 2008. ACM.

[de Haan 09a] Gerwin de Haan & Frits H. Post. *StateStream: a Developer-Centric Approach Towards Unifying Interaction Models and Architecture*. In Proc. of ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS 2009, pages 13–22, July 2009.

[de Haan 09b] Gerwin de Haan, Josef Scheuer, Raymond de Vries & Frits H. Post. *Egocentric Navigation for Video Surveillance in 3D Virtual Environments*. In 3D User Interfaces, 2009. 3DUI '09. IEEE Symposium on, Lafayette, LA,, March 2009.

[Han 05] Jefferson Y. Han. *Low-cost multi-touch sensing through frustrated total internal reflection*. In UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology, pages 115–118, New York, NY, USA, 2005. ACM.

[Harel 87] David Harel. *Statecharts: A visual formalism for complex systems*. Sci. Comput. Program., vol. 8, no. 3, pages 231–274, 1987.

[Hartmann 06] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher & J. Gee. *Reflective physical prototyping through integrated design, test, and analysis*. Proc. of UIST '06, pages 299–308, 2006.

[Heldal 05a] Ilona Heldal, Anthony Steed, Ann-Sofie Axelsson & Josef Wideström. *Immersiveness and Symmetry in Copresent Scenarios*. In Proc. IEEE VR '05, pages 171–178. IEEE Computer Society, 2005.

[Heldal 05b] Ilona Heldal, Anthony Steed & Ralph Schroeder. *Evaluating Collaboration in Distributed Virtual Environments for a Puzzle-solving Task*. In Proc. HCI International, 2005.

[Hendricks 03] Zayd Hendricks, Gary Marsden & Edwin Blake. *A meta-authoring tool for specifying interactions in virtual reality environments*. In Proc. of AFRIGRAPH '03, pages 171–180, 2003.

[Hentschel 08] Bernd Hentschel, Irene Tedjo, Markus Probst, Marc Wolter, Marek Behr, Christian Bischof & Torsten Kuhlen. *Interactive Blood Damage Analysis for Ventricular Assist Devices*. IEEE Transactions on Visualization and Computer Graphics, vol. 14, no. 6, pages 1515–1522, 2008.

[Hultquist 92] J. P. M. Hultquist & E. L. Raible. *SuperGlue: a programming environment for scientific visualization*. In Proc. IEEE Visualization '92, pages 243–250, Boston, MA, USA, 1992.

[Jacob 99] R. J. K. Jacob, L. Deligiannidis & S. Morrison. *A software model and specification language for non-WIMP user interfaces*. Transactions on Computer-Human Interaction, vol. 6, no. 1, pages 1–46, 1999.

[Johnson 04] Christopher Johnson & Charles Hansen. Visualization Handbook. Academic Press, Inc., Orlando, FL, USA, 2004.

[Koutek 01a] M. Koutek & F.H. Post. *A Software Environment for the Responsive Workbench*. In R.L. Lagendijk & J.W.J. Heijnsdijk, editors, Proc. ASCI '01, pages 428–435. ASCI, Netherlands, 2001.

[Koutek 01b] M. Koutek & F.H. Post. *Dynamics Manipulation Tools for the Responsive Workbench*. In M. Hirose & H. Tamura, editors, Proc. International Symposium on Mixed Reality '01, pages 167–168. University of Tokyo, Japan, 2001.

[Koutek 01c] M. Koutek & F.H. Post. *Spring-Based Manipulation Tools for Virtual Environments*. In B. Fröhlich, J. Deisinger & H.-J. Bullinger, editors, Immersive Projection Technology and Virtual Environments, proceedings of the Eurographics Workshop, pages 61–70, May 2001.

[Koutek 02] M. Koutek, J. van Hees, F.H. Post & A.F. Bakker. *Virtual Spring Manipulators for the Particle Steering in Molecular Dynamics on the Responsive Workbench*. In Proc. Eurographics Virtual Environments '02, pages 55–62, 2002.

[Koutek 03] Michal Koutek. *Scientific Visualization in Virtual Reality: Interaction Techniques and Application Development*. PhD thesis, Delft University of Technology, 2003.

[Koutek 07] Michal Koutek, René Molenaar, Gerwin de Haan & Frits H. Post. *Visual Consistency in Rotational Manipulation Tasks in Sheared-Perceived Virtual Environments*. In Proc. of Immersive Projection Technology and Eurographics Virtual Environments '07, july 2007.

[Krüger 95] Wolfgang Krüger, Christian-A. Bohn, Bernd Fröhlich, Heinrich Schüth, Wolfgang Strauss & Gerold Wesche. *The Responsive Workbench: A Virtual Work Environment*. Computer, vol. 28, no. 7, pages 42–48, 1995.

[Larimer 03] Daniel Larimer & Doug A. Bowman. *VEWL: A Framework for Building a Windowing Interface in a Virtual Environment*. In Proceedings of INTERACT: IFIP TC13 International Conference on Human-Computer Interaction, pages 809–812, 2003.

[Latoschik 08] Marc Eric Latoschik, Dirk Reiners, Roland Blach, Pablo Figueroa & Raimund Dachselt. IEEE VR 2008 Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS). Shaker Verlag, Aachen, Germany, 2008.

[Liang 94] J. Liang & M. Green. *JDCAD: A highly interactive 3D modeling system*. Computers & Graphics, vol. 18, no. 4, pages 499–506, 1994.

[Lindeman 99a] Robert W. Lindeman, John L. Sibert & James K. Hahn. *Hand-Held Windows: Towards Effective 2D Interaction in Immersive Virtual Environments*. In Proceedings of IEEE VR, page 205, 1999.

[Lindeman 99b] Robert W. Lindeman, John L. Sibert & James K. Hahn. *Towards usable VR: an empirical study of user interfaces for immersive virtual environments*. In Proc. CHI '99, pages 64–71, 1999.

[Lindeman 01] Robert W. Lindeman, John L. Sibert & James N. Templeman. *The Effect of 3D Widget Representations and Simulated Surface Constraints on Interaction in Virtual Environments*. In Proceedings of IEEE VR 2001, pages 141–148, 2001.

[Lüttgen 00] Gerald Lüttgen, Michael von der Beeck & Rance Cleaveland. *A compositional approach to statecharts semantics*. SIGSOFT Softw. Eng. Notes, vol. 25, no. 6, pages 120–129, 2000.

[Massink 99] Mieke Massink, David J. Duke & Shamus P. Smith. *Towards Hybrid Interface Specifications for Virtual Environments*. In Proc. of DSV-IS, pages 30–51, 1999.

[Molenaar 07] René Molenaar. Viewing and Interaction for Multiple Users in Projection-Based VR Systems. Master's thesis, Delft University of Technology, Data Visualization Group, april 2007.

[Mulder 02] J.D. Mulder & R. van Liere. *Personal Space Station*. In Proc. VRIC '02, pages 73–81. Laval Virtual, 2002.

[Munzner 06] Tamara Munzner, Chris Johnson, Robert Moorhead, Hanspeter Pfister, Penny Rheingans & Terry S. Yoo. *NIH-NSF Visualization Research Challenges Report Summary*. IEEE Comput. Graph. Appl., vol. 26, no. 2, pages 20–24, 2006.

[Myers 00] Brad Myers, Scott E. Hudson & Randy Pausch. *Past, present, and future of user interface software tools*. ACM Transactions Computer-Human Interaction, vol. 7, no. 1, pages 3–28, 2000.

[Olsen 07] Dan R. Olsen. *Evaluating user interface systems research*. In Proc. of UIST '07, pages 251–258, 2007.

[Olwal 04] Alex Olwal & Steven Feiner. *Unit: modular development of distributed interaction techniques for highly interactive user interfaces*. In Proc. of GRAPHITE '04, pages 131–138, 2004.

[Osawa 05] N. Osawa. *Enhanced Hand Manipulation for Efficient and Precise Positioning and Release*. In Erik Kjems & Roland Blach, editors, Proceedings of the 9th IPT and 11th Eurographics VE Workshop (EGVE) '05, pages 221–222, October 2005.

[Ousterhout 98] J. K. Ousterhout. *Scripting: higher level programming for the 21st Century*. Computer, vol. 31, no. 3, pages 23–30, 1998.

[van de Pol 99] R. van de Pol, W. Ribarsky, L. Hodges & F.H. Post. *Interaction Techniques on the Virtual Workbench*. In Proc. Eurographics Virtual Environments '99, pages 157–168, 1999.

[Ponder 03] M. Ponder, G. Papagiannakis, T. Molet, N. Magnenat-Thalmann & D. Thalmann. *VHD++ development framework: towards extendible, component based VR/AR simulation engine featuring advanced virtual character technologies*. 2003. Proceedings Computer Graphics International, pages 96–104, 2003.

[Ray 07] Andrew Ray & Doug A. Bowman. *Towards a system for reusable 3D interaction techniques*. In Proc. of VRST '07, pages 187–190, 2007.

[Reiners 02] Dirk Reiners, Gerrit Voß& Johannes Behr. *OpenSG: Basic Concepts*. In Proc. OpenSG Symposium, 2002.

[Rohlf 94] John Rohlf & James Helman. *IRIS performer: a high performance multiprocessing toolkit for real-time 3D graphics*. In Proc. SIGGRAPH '94, pages 381–394, 1994.

[Scheidegger 08] Carlos E. Scheidegger, Huy T. Vo, David Koop, Juliana Freire & Claudio T. Silva. *Querying and re-using workflows with VisTrails*. In Proc. SIGMOD '08, pages 1251–1254, New York, NY, USA, 2008. ACM.

[Schmalstieg 99] D. Schmalstieg, L.M. Encarnacao & Z. Szalavari. *Using Transparent Props for Interaction With The Virtual Table*. In Proc. ACM Symp. Interactive 3D Graphics '99, pages 147–154, 1999.

[Schroeder 06] Will Schroeder, Ken Martin & Bill Lorensen. The Visualization Toolkit, Third Edition. Kitware Inc., 2006.

[Shaer 05] O. Shaer & R. J. K. Jacob. *Toward a Software Model and a Specification Language for Next-Generation User Interfaces*. CHI '05 Workshop on The Future of User Interface Software Tools, 2005.

[Shan 08] Jie Shan & Charles K. Toth. Topographic Laser Ranging and Scanning: Principles and Processing. CRC Press, 2008.

[Shreiner 05] D. Shreiner, M. Woo, J. Neider & T. Davis. OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition). Addison-Wesley Professional, August 2005.

[Simon 05] Andreas Simon. *First-person experience and usability of co-located interaction in a projection-based virtual environment*. In VRST '05: Proceedings of the ACM symposium on Virtual reality software and technology, pages 23–30, New York, NY, USA, 2005. ACM Press.

[Simon 07] Andreas Simon. *Usability of Multiviewpoint Images for Spatial Interaction in Projection-Based Display Systems*. IEEE Trans. Vis. Comput. Graph., vol. 13, no. 1, pages 26–33, 2007.

[Smit 08] F. A. Smit, R. van Liere & B. Fröhlich. *An image-warping VR-architecture: design, implementation and applications*. In Proc. VRST '08, pages 115–122, 2008.

[Smith 07] Shamus P. Smith. *Exploring the Specification of Haptic Interaction*. In Interactive Systems. Design, Specification, and Verification, volume 4323 of *Lecture Notes in Computer Science*, pages 171–184, 2007.

[Springer 00] Jan Springer, H. Tramberend & Bernd Fröhlich. *On Scripting in Distributed Virtual Environments*. In Proceedings of the 4th IPT Workshop, June 2000.

[Stanton 04] Eric T. Stanton & W. Philip Kegelmeyer. *Creating and Managing "Lookmarks" in ParaView*. In Proc. INFOVIS '04, page 215.19, Washington, DC, USA, 2004. IEEE Computer Society.

[Steed 04] A. Steed & C. Parker. *3D Selection Strategies for Head Tracked and Non-Head Tracked Operation of Spatially Immersive Displays*. In 8th International Immersive Projection Technology Workshop(IPT2004), 2004.

[Steed 06] Anthony Steed. *Towards a General Model for Selection in Virtual Environments*. In Proc. IEEE 3DUI '06, pages 103–110, 2006.

[Steinicke 05] F. Steinicke, T. Ropinski & K. Hinrichs. *VR and Laser-Based Interaction in Virtual Environments Using a Dual-Purpose Interaction Metaphor*. In In IEEE VR 2005 Workshop Proceedings on New Directions in 3D User Interfaces, pages 61–64, 2005.

[Stolk 02] Bram Stolk, Faizal Abdoelrahman, Anton Koning, Paul Wielinga, Jean-Marc Neefs, Andrew Stubbs, An de Bondt, Peter Leemans & Peter van der Spek. *Mining the human genome using virtual reality*. In Proc. EGPGV '02, pages 17–21, 2002.

[Strauss 93] Paul S. Strauss. *IRIS Inventor, a 3D graphics toolkit*. In Proc. OOPSLA '93, pages 192–200, 1993.

[Szalavári 97] Zs. Szalavári & M. Gervautz. *The Personal Interaction Panel: A Two-Handed Interface for Augmented Reality*. In Computer Graphics Forum 16(3), pages 335–346, 1997.

[Teylingen 97] Ron van Teylingen, William Ribarsky & Charles van der Mast. *Virtual Data Visualizer*. Transactions on VIsualization and Computer Graphics, vol. 3, no. 1, pages 65–74, 1997.

[Tramberend 99] Henrik Tramberend. *Avocado: A Distributed Virtual Reality Framework*. In Proc. IEEE VR '99, pages 14–21, 1999.

[Upson 89] Craig Upson, Thomas Faulhaber Jr., David Kamins, David H. Laidlaw, David Schlegel, Jefrey Vroom, Robert Gurwitz & Andries van Dam. *The Application Visualization System: A Computational Environment for Scientific Visualization*. IEEE Comput. Graph. Appl., vol. 9, no. 4, pages 30–42, 1989.

[van Dam 00] A. van Dam, A. S. Forsberg, D. H. Laidlaw, J. LaViola & R. M. Simpson. *Immersive VR for Scientific Visualization: A Progress Report*. IEEE Computer Graphics and Applications, pages 26–52, Nov/Dec 2000.

[Vanacken 06] Davy Vanacken, Joan De Boeck, Chris Raymaekers & Karin Coninx. *NIMMIT: A notation for modeling multimodal interaction techniques*. In Proc. of GRAPP, pages 224–231, 2006.

[Wartell 99] Zachary Justin Wartell, Larry F. Hodges & William Ribarsky. *The Analytic Distortion Induced by False-Eye Separation in Head-Tracked Stereoscopic Displays*. Technical report, Georgia Tech, GVU Lab, 1999.

[Watsen 98] K. Watsen & M. Zyda. *Bamboo - A Portable System for Dynamically Extensible, Real-Time, Networked, Virtual Environments*. In VRAIS '98: Proceedings of the Virtual Reality Annual International Symposium, page 252, Washington, DC, USA, 1998. IEEE Computer Society.

[Watsen 99] Kent Watsen, Rudolph P. Darken & Michael Capps. *A Handheld Computer as an Interaction Device to a Virtual Environment*. In Proceedings of the International Projection Technologies Workshop, 1999.

[Wernecke 93] Josie Wernecke. The Inventor Mentor: Programming Object-Oriented 3d Graphics with Open Inventor, Release 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[Wetzstein 07] Gordon Wetzstein, Moritz Göllner, Stephan Beck, Felix Weiszig, Sebastian Derkau, Jan P. Springer & Bernd Fröhlich. *HECTOR - scripting-based VR system design*. In SIGGRAPH '07: ACM SIGGRAPH 2007 posters, page 143, New York, NY, USA, 2007. ACM.

[Wingrave 01] C. Wingrave, D. Bowman & N. Ramakrishnan. *A First Step Towards Nuance-Oriented Interfaces for Virtual Environments*. In Proc. of the Virtual Reality International Conference, pages 181–188, 2001.

[Wingrave 05] C. Wingrave & D. Bowman. *"CHASM": Bridging Description and Implementation of 3D Interfaces*. In Proc. of IEEE VR Workshop on New Directions in 3DUIs, pages 85–88, 2005.

[Wingrave 08] C. Wingrave & D. Bowman. *Tiered Developer-Centric Representations for 3D Interfaces: Concept-Oriented Design in Chasm*. In Proc. of IEEE VR '08, pages 193–200, 2008.

[x3d 07] X3D: Extensible 3D Graphics for Web Authors (The Morgan Kaufmann Series in Interactive 3D Technology). Morgan Kaufmann, April 2007.

[Zachmann 96] G. Zachmann. *A language for describing behavior of and interaction with virtual worlds*. In Proc. of VRST '96, pages 143–150, 1996.

Summary

Techniques and Architectures for 3D Interaction

Gerwin de Haan

Spatial scientific datasets are all around us, and 3D visualization is a powerful tool to explore details and structures within them. When dealing with complex spatial structures, interactive Virtual Reality (VR) systems can potentially improve exploration over desktop-based systems. However, from previous experience it was found that traditional interaction tools as used in desktop visualization systems do not function well within a Virtual Environment (VE), especially not for complex spatial data exploration tasks. Also the application of basic 3D interaction techniques is not straightforward, as they require customization for specific scenarios, applications and VR systems. This process of design, development and customization of 3D interaction techniques is not well supported by existing software tools for VR and visualization.

In this thesis we focus on enhancing 3D interaction techniques and interfaces for data exploration in VR. Based on observations from human actions and perception in tracked, projection-based VR systems, we extend some fundamental 3D interaction metaphors. We propose an enhanced technique for 3D selection of objects in a VE, and explore 3D selection and manipulation in collaborative situations.

From our work on these techniques we have come to consider 3D interaction as a complex, first-class component of 3D systems. In the integration of these advanced techniques in our framework and applications, we experienced many iterations of redesign and tuning. To better support this emergent development style, we developed a prototyping architecture for constructing and adapting VR applications. Because of the complexity of 3D interaction techniques, further support for their specific design and development was required. We extended the prototyping architecture with a modelling approach, specific to the description, execution and iterative improvement of interaction techniques.

To truly enable the potential of an interactive VR system, expertise from many areas such as perception, usability design and software engineering needs to be integrated in its design. With the contributions presented in this thesis, we hope to improve this view and facilitate prototypical design and development cycles for improved 3D environments for interactive exploration.

Samenvatting

Technieken and Architecturen voor 3D Interactie

Gerwin de Haan

Wetenschappelijke simulatie- en acquisitietechnieken genereren omvangrijke multidimensionale datasets, welke met behulp van 3D visualisatietechnieken geanalyseerd kunnen worden. Vooral waar het gaat om het bestuderen van complexe, ruimtelijke structuren, kunnen interactieve Virtual Reality (VR) systemen extra mogelijkheden bieden boven 2D desktop gebaseerde systemen. Binnen een VR omgeving zijn standaard, muis-gebaseerde 2D interactietechnieken niet toereikend. Goed gebruik van bestaande 3D interactietechnieken blijkt ook beperkt, en deze behoeven deze veelvuldig aanpassing voor specifieke scenario's, toepassingen en VR systemen. Dit proces van ontwerp, ontwikkeling en aanpassing van 3D interactietechnieken blijkt niet goed te worden ondersteund door bestaande software omgevingen voor VR en visualisatie.

Gebaseerd op observaties van wat haalbaar is binnen *getrackte*, projectie-gebaseerde VR systemen, stellen we verbeterde 3D object selectietechnieken voor die meer rekening houden met de onnauwkeurigheden in de menselijke perceptie en motoriek. Ook stellen we 3D interactietechnieken voor die toegespitst zijn voor situaties waarin meerdere gebruikers gezamenlijk werken binnen een VR systeem.

Tijdens de ontwikkeling van deze 3D interactietechnieken bleek dat, door hun complexiteit, vele ontwerp- en ontwikkeliteraties benodigd waren voordat deze goed werkten. Hierdoor zijn we de ontwikkeling van 3D interactietechnieken gaan beschouwen als een zeer belangrijk en wat ondergewaardeerd onderdeel van bestaande VR en visualisatiesystemen. Om beter aan deze iteratieve stijl van ontwikkeling tegemoet te komen, stellen we een ontwikkelaanpak voor die de beschrijving en implementatieslagen van interactietechnieken ondersteunt. Deze is gebaseerd op een *prototyping* architectuur voor VR applicaties, die verschillende abstractielagen flexibel combineert met het gebruik van een scriptingtaal.

Om de mogelijkheden van een interactief VR-systeem effectief te benutten, zal expertise op gebied van perceptie, *usability design* en *software-engineering* moeten worden geïntegreerd in het ontwerpproces. Met de bijdragen gepresenteerd in dit proefschrift hopen we dit standpunt te versterken en een basis te hebben gelegd voor een verdere ontwikkeling van verbeterde 3D omgevingen voor interactieve exploratie.

Curriculum Vitae

Gerwin de Haan was born on September 7th, 1977 in Rotterdam, The Netherlands. In 1995, he completed secondary school at the *RSG Hoekse Waard* in Oud-Beijerland. He started studying Computer Science at Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) and received his Master's degree in 2002. He performed his thesis work in the Computer Graphics and CAD/CAM group. The thesis work focussed on integrating scientific simulation software in Virtual Environments and was titled *"Interactive Visualization on the Virtual Reality Workbench"*.

In the following two years, Gerwin continued work on scientific simulations at *Foldyne*, a life-sciences spin-off company of the Faculty of Applied Sciences of Delft University of Technology. His work there focussed on Molecular Dynamics simulation software. He also performed technical development on several interactive and graphical web-based projects, both at the company *Mindbus* and as a freelance software developer under the name *Extensible*.

In 2004, Gerwin returned to the Computer Graphics group in Delft to pursue a PhD on the subject of 3D interaction in Virtual Reality. Aside from his research activities, he was actively involved in the development of hard- and software for the new VR Laboratory facilities. His teaching activities include assistance with several Master's and PhD level courses and practical exercises on the subject of computer graphics, visualization and Virtual Reality. In addition, he supervised several Master's and Bachelor's student projects, often in collaboration with external partners in industry. His PhD research period was paused for a year in 2008, which he spent working on the topic of visualization 3D topographical point-cloud data at the OTB Institute in Delft. In the spring of 2009 he completed his PhD project and this dissertation.

Acknowledgements

After two years of working in industry I was offered this PhD position in the area of interaction and Virtual Reality. I felt this was a great opportunity to work on VR again and to get into academia. What really convinced me was that I would not start out all by myself, nor empty handed: I would be collaborating with new colleagues on Virtual Reality and funding was available for new and exciting equipment for the VR laboratory. Looking back after some years, I am really glad that I took this opportunity and I am still excited to continue working in this area. I would like to acknowledge some, if not all of the people who were somehow involved in this project.

First I would like to thank my advisors Frits Post, copromotor, and Erik Jansen, promotor. Frits, thank you for all the valuable and entertaining discussions we had almost on a daily basis. These, combined with your written feedback and suggestions, provided invaluable insights on our research field and far beyond. Erik, I really value the speed and effectiveness of the feedback you have given me on all of my work, and especially on this thesis.

Working in a small VR team was essential for keeping work as fun and interesting as it was. Eric Griffith was my direct colleague and a great "partner in crime". Eric, I envy your productivity and skills both in computer science and in organizing social activities. I hope you stay around so we have the opportunity to work with you and have more special beers with you in future. Michal Koutek was my former MSc supervisor and a good sparring partner during my PhD project. Michal, we often disagreed and Eric and I sometimes gave you a hard time. However, your positivism, persistence and help in making things work never stopped to surprise me. I wish you and your family the best of luck with your new start at KNMI on the higher grounds of Utrecht.

My colleagues in the Computer Graphics group were always there for inspirational and entertaining discussions during coffee breaks, lunch, Frisbee sessions, colloquia and "sugar fixes". I won't list you all, but I would like to specially thank those with whom I directly collaborated in some way: Rafael Bidarra, Jorik Blaas, Charl Botha, Stef Busking, Peter Krekel and Lingxiao Zhao. Although the subject of Virtual Reality has often been subject of ridicule, you were always willing to give practical

advise. Without the support of Ruud de Jong and Bart Vastenhouw I would still be stuck on an old Debian machine, I thank you for that. I am also thankful for the support of Toos Brussee-Donders and Bianca Abrahamsz, without you I would still be waiting for a reply on travel expenses from some service point. Our colleagues from Applied Physics were helpful in keeping things up and running in our Virtual Reality Lab, and they were great for discussing matters of physics with, of course. I thank Loek Bakker, Jaap Flohil, Harm Jonker, Thijs Heus, Erwin de Beus, Angela de Ceuninck van Capelle and Joyce van Velzen for their support.

I also would like to thank the students whom I got to work with. With all of you it was really inspiring to experience the transition from early stage supervision to the eventual side-by-side collaboration. The students include the ones I supervised directly, in chronological order: Ruixing Wang, Rene Molenaar, Josef Scheuer and Huib Piguillet. Also I thank the students in whose projects I got somehow indirectly involved in, namely Mark van Ameijden, Dylan Dussel, Torsten Stöter and the "Wii-team" at TNO. I hope I did not push you too hard in my excitement.

I thank external colleagues for some extensive discussions I had with them on VR, visualisation or interaction: Stijn Oomes (Man-Machine Interaction), Aad-Jan van der Helm, Jouke Verlinden (Industrial Design) Michel van Elk, Raymond de Vries (TNO Delft), Arjen van Rhijn, Jurriaan Mulder, Robert van Liere, Ferdi Smit (CWI Amsterdam), Jorrit Adriaanse, Bram Stolk (SARA Amsterdam), Lode Vanacken (EDM Hasselt), the VR-Vis team at RWTH Aachen, Anthony Steed (UCL London) and Oliver Kreylos (USC Davis).

My family and friends have always been supportive of my work. I would like to take the opportunity to specially thank my parents for doing a good job of supporting me in sports and in education. Also I thank my former flatmates Yuri Martens and Robert Grauss for dealing with nerdy me during the first two years of my project. Yuri, I wish you all the best with your new PhD adventures in London. Robert, I hope your skills in research and micro surgery won't be lost in the daily life of the hospital.

Finally, I thank Blanca for supporting me and coping with me. Especially in the last year I could not join her on as many parties, concerts and trips as we would have liked to. Still, she proved to be a great motivator, also by giving me my own little room in which I could work on this thesis. Blanca, although we unfortunately did not make it to the defence as a couple, I am ever grateful to you for your support through the years.