Interactive Exploration in Virtual Environments

Interactive Exploration in Virtual Environments

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. mr. P.F. van der Heijden
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 23 april 2003, te 14:00 uur

door

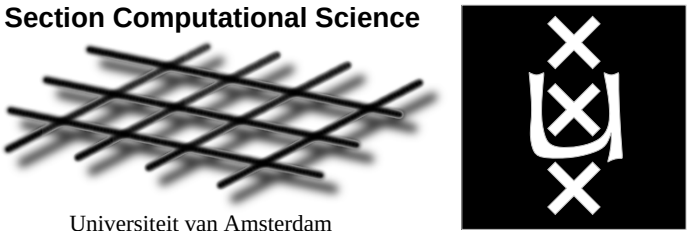
Robert Gerardus Belleman

geboren te Egmond aan Zee

Promotor:	prof. dr. P.M.A. Sloot
Commissie:	prof. dr. ir. F.C.A. Groen prof. dr. ir. J.H.C. Reiber prof. dr. ir. H.E. Bal prof. drs. M. Boasson prof. dr. A.V. Bogdanov dr. P. van Emde Boas dr. G.D. van Albada
Faculteit:	Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The work described in this thesis was financially supported by the Netherlands Organization for Scientific Research (NWO) under grant 612-021-103, SARA Computing and Networking Services, and Silicon Graphics, Inc.

The work described in this thesis has been carried out at the Section Computational Science of the University of Amsterdam.



ISBN 90-5776-101-7

Typeset with LaTeX 2ε.
Printed by Ponsen & Looijen, Wageningen.
Author contact: `robbel@bijmij.net`

On the cover: The cover appears to be a random collection of shapes, but in fact, it contains more. Relax your eyes and stare *through* the cover. At some point you should see a pattern jump out of the page. The solution is drawn on this page.

Contents

Acknowledgments

The work described in this thesis would not have been possible without the help of many others.

First and foremost; I would like to thank my promotor, *Peter Sloot*. He was one of the key figures in bringing Virtual Reality to Amsterdam and by doing so, he gave great impulse to Virtual Reality research in the Netherlands. He knew I had a great interest in this area and gave me the opportunity, freedom and support to work on what now lies before you. Peter, many thanks for all your help, both in bad and good times, in research and other matters.

Thanks to *Jaap Kaandorp* who supervised my first steps in this research. One of the first CAVE applications we built was, of course, for the visualization of Jaap's corals. I think that around that time he still read his email using `more(1)`. Jaap's contacts with radiologists in different hospitals were my first introduction to DICOM which proved very helpful later on in my research.

I hold a deep respect for *Dick van Albada* who, besides knowing a lot about everything, is always friendly and willing to patiently answer questions despite the ever growing piles of paperwork in his office. I'm also very grateful for his scrutinous reading of this thesis.

Many thanks to *Alfons*, *Artoli* and *Roeland* who were always prepared to enlighten me on the subtle intricacies of the lattice Boltzmann algorithm.

I know he doesn't like it when I say this, but *Roman Shulakov* is as Russian as a man can get. His deep, rumbling voice as he enters my room while saying, "Rob... I have a trouble..." always puts a smile on my face. Roman did most of the work in implementing the CAVERNUS based network communication pipeline.

Denis Shamonin turned out to be a talented code writer. Some of his code is now used by people all over the globe and I'm quite sure more will follow soon.

I must have done something wrong that makes *Elena Zudilova* always stop at the door porch and look apologizing before entering my room to ask me something. Whatever it was, I suppose one of the reasons must be that she now has to put up with my

software that invariably misinterprets her voice commands. I swear I didn't put that in there intentionally!

Zhiming, our HLA guru, was instrumental for doing the measurements on HLA's performance and setting me straight on things I thought I knew about HLA, but didn't.

During the last months of my work I shared my office with *Simon*, a talented guy who radiates enthusiasm toward his research. Are you still serious about letting students solve the traveling salesman problem between distant moving stars in a star cluster?

All you other boys and girl at the SCS; astro girl *Alessia*, party animal *Alfredo*, Nova Zembla neighbour *David*, resource manager *Judhi*, MPI wizard *Kamil*, GRAPE *Piero*, best teacher award winner *Walter*: I wish you all the best with your work.

The work on simulated vascular reconstruction was sparked by an inspiring presentation given by *Charles Taylor*, Stanford University, at the *Medicine Meets Virtual Reality* conference in San Francisco, 1999. This resulted in a visit by *Sean Spicer*, a student at Taylor's research group, to the University of Amsterdam. Sean developed the software that allowed us to use OpenGL|Volumizer in CAVE applications and he implemented the first Virtual Reality version of a surgical planning system. Sean's work was influential in our VRE environment and later in the virtual operating theatre for simulated vascular reconstruction.

The VRE project was a collaboration between the University of Amsterdam, Leiden University Medical Center, LKEB, IBM, Medis B.V., SARA and Arcobel. I am greatly indebted to all the people who have invested their time in showing me around their departments, discussing their work and criticizing the intentions of VRE. In particular, I would like to thank *Hans Reiber* for organizing my visit to the radiology department of the Leiden University Medical Center (LUMC) and *Bart ter Haar Romeny* for organizing my visit to the radiology department of the University Medical Center Utrecht (UMC). Also many thanks to all at the LUMC and LKEB for their help, insights and ideas; *Jorrit Schaap*, *Rob van der Geest*, *Patrick Koning*, *Kees Verlooij*, *Aart Spilt* and *Mark van Buchem*.

The work on *Madracis* was done with *Mark J.A. Vermeij* (Institute for Biodiversity and Ecosystem Dynamics, University of Amsterdam), *Rolf P.M. Bak* (Netherlands Institute of Sea Research), *Leo E.H. Lampmann* (St. Elisabeth Hospital, Tilburg). *Dick Bakker*, radiologist at the Leiden University Medical Center scanned the *Pocillopora damicornis* corals used in the work described in section 2.4.

Thanks to *Maarten Boasson* and *Erik Boasson* for providing us with a version of SPLICE and their willingness to explain and discuss its details.

I have been in the fortunate position to have had talented students who have con-

tributed to this research with their projects.

Daniel Fontijne (formerly Dijkman) did most of the programming, as a student project, for GEOPROVE. On the side, he implemented a Windows version of the CAVE library, just because he wanted to do his software development on his PC at home instead of on an SGI O2 in our lab. This software almost got him into legal trouble when he wanted to give his code away for free. It took VRCO another four years to release their first Windows version. I rest my case.

Don Hannema took my rudimentary version of an interaction library to use Vtk in the CAVE and transformed it into SCAVI. This code is the foundation of the virtual operating theatre for simulated vascular reconstruction.

Hans Ragas implemented the flow visualization code used for this same environment. This was by no means an easy task, but he succeeded nevertheless. I wonder when he will commit the bug fix into the CVS repository that repairs the colour table bug that is still in there.

I would also like to thank *Harro Mantel*, *Frank Lakeman*, *Hongjing Wu*, *Bart van der Schans*, *Anton Hendriks* and *David Vismans* for their valuable contributions.

At times, I must have driven the people at SARA stark raving mad with all my questions, phone calls and (sometimes not too friendly) emails to "hic" (or is it "hec"?) regarding the O2 in our lab, DCE/DFS, the IBM SP2, HiPPI, the cluster, Teras and countless CAVE reservations and other questions.

Anton Koning always proved extremely helpful in providing solutions to problems that boggled my mind. There have been cases in which his help took nothing more than the magical touch of just one button... literally. He generously allowed us to use his code in our software, such as the menu system and transfer table editor used in VRE, and SARAnav, which I think is now *the* most often used piece of software by anyone who has access to a CAVE. But Anton; don't you think it's about time you stopped putting thousands of lines of code in a single source file?

Bram Stolk generously gave us his implementation of a speech recognition system which we extended to support context sensitivity and speech synthesis. Bram was helpful with many issues concerning VR and can truely be called an "expert". His latest work on the exploration of Human Genome data has recently been published in Science; see [20]. Together with *Raymond de Vries* we experimented with the PC/Linux based VR systems and wrote a paper on it. So guys; when do you think we can build an XBox/Linux based VR system?

I owe *Paul Wielinga*, *Bas van der Vlies*, *Axel Berg*, *Wilfred Janssen*, *Jaap Dijkshoorn*, *Edward Breedveld*, *Arnaud Leijen*, *Alexander Verkooijen*, *Jeroen van Hoof*, *Robbert*

van Dale, *Willem Vermin*, *Jules Wolfrat* and probably many more that have been shielded from me a great deal of thanks for all they have done.

I would like to thank my thesis reading committee for taking part in my PhD defense and for their comments on the manuscript; *Frans Groen*, *Hans Reiber*, *Henri Bal*, *Maarten Boasson*, *Alexander Bogdanov*, *Peter van Emde Boas* and *Dick van Albada*.

Of course, I will not forget all the people at the secretariat for all their help; *Erik*, *Jackie*, *Marianne*, *Virginie*, *Saskia*, *Hugo* and also *Laura* and *Ina*.

System management at the Kruislaan runs a tight ship, there is no denying that. We have regularly had, and still have, our differences of opinion, but these guys are always prepared to listen to my grievances and come up with a solution in the end. *Gert*, *Jan W.*, *Robbert*, *Ruud*, *Gerard*, *Frans L.*, *Frans H.*, *Stephan*, *Cees*, *Ari*, *Derk* and *Adri*; I thank you.

I miss the drinks and many laughs I had with the "old" guys that have left the UvA: *Arjen*, *Benno*, *Diederik*, *Drona*, *Jan*, *Jeroen*, *Martin*, *Joep*, *Frank* and yes; also *Berry*. I'm glad most of us still get together sometimes to remember the good times.

I'm very much looking forward to working with all the people involved in recently initiated and future projects; of course *Bob Hertzberger* for giving me this unique opportunity. Past, present and future Virtual Laboratory people; *Hakan*, *Hamideh*, *Adam*, *Ercin*, *Ammar*, *David*, *Cees*, *Zeger*, *Toto*, *Vladimir*, *Dmitry*, *Cesar*, *Anne*. The People at Philips NatLab, in particular *Henk Obbink*, *Jaap van der Heijden* and all others at the SWA department. Many thanks to *Ronald Schut* for securing my financial situation in such short notice.

Many thanks to my friends and family for their encouragement, in particular *mam en pap*, *Jeroen*, *Joëlle*, *Saskia*, *Rob*, *José*, *Ron*, *Durk* and of course *Ben*, *Ramin*, *Bas* and *Tobias*. This would have been the first performance as "paranimfen" for Saskia and Ben. However, Ben had to cancel at the last moment because of a far more important and happier occasion; the arrival of his and Marije's second adopted daughter. I'm very happy that Ramin has agreed to take over.

Words cannot express what *Eva* means to me, so I won't try. But, after having made the final changes to this thesis, I suppose there is just one last thing that I must do, and that is to live up to my promise to her: to quit smoking when my thesis is finished.

Well; it's done.

I quit.

Robert Belleman,
Amsterdam, February 2003.

Chapter 1

Interactive exploration environments

"Computers are useless. They can only give you answers."

Pablo Picasso (1881-1973).

1.1 Introduction

The increase in availability of computational resources, both storage capacity and processing speed, have allowed researchers in industrial and scientific areas to investigate increasingly large and complex time dependent problems. As a result, the data sets that are generated by these applications grow larger and more complex. Furthermore, many of the industrial and scientific applications are typically simulations of complex systems. In general these problems are intractable and *NP* complete so that the only available option to obtain insight in these problems is through explicit simulation. As in this class of problems the parameter space is typically extremely large, it is not feasible to simulate every point in the parameter space. Optimization algorithms that perform a guided search through the problem's parameter space have been used to avoid this [221], still, as these parameter spaces grow larger, the time that is required to find a satisfactory solution is unacceptable.
In many cases, the automated analysis of these data spaces is difficult, either because no techniques are available to extract the features of interest or otherwise simply because it is unknown what information is present in the data beforehand. Interactive exploration may be one of the few options to analyse this data in order to obtain further insight in these data spaces.

1.2 Interactive exploration

Often, the only alternative to obtain insight in large, complex data sets in cases where the automatic analysis of data spaces is impossible is through human inspection. Here, a human being takes on the role of analyst who uses his knowledge to analyse

the data. Unfortunately, the increase in computing capacity is not paralleled by an equivalent increase in human capacity. Hardly ever will the analyst resolve to inspection methods where he scrutinizes piles of numbers in search of important clues since this is cumbersome, prone to subjective errors and not in the least, mind numbing. Instead, the data is converted into a representation that allows the analyst to "explore" the data and perceive patterns that will help him find structure. If performed correctly, the analyst's senses, cognitive abilities and experience together will help him in forming a mental picture that leads to a better understanding of the data. The primary goal of exploration therefore is to increase the bandwidth to the brain.
The challenges in creating an environment that is suitable for this type of exploration are many. First, the environment should be able to represent data in a correct, clear and informative manner. The conversion of data into another representation leads to the risk of introducing patterns in data that really are not there, so care must be taken in which methods are used. Indeed, it is often a good idea to provide multiple methods so that the user has the ability to interactively choose between different representations of the same data. Second; the conversion of data into another representation inevitably takes time. The amount of time should be minimized in order to obtain an environment that is responsive to user interaction. Third; exploration environments will only be effective if the explorer does not feel restricted by the environment during his work. The user should be allowed to interact with the different aspects of his experiment in an intuitive manner. In addition, the environment should provide clear and rapid feedback in response to user interaction.

Unfortunately, we will see that these different problems are hard to solve all at once. As each design choice has its implications on the performance of another we will see that compromises are often necessary. In this work we distinguish two types of environments; static and dynamic exploration environments.

1.2.1 Static environments

In Interactive Static Exploration Environments (ISEE), the data presented to the user is time invariant. The data that is to be explored is generated by an external process, such as a computer simulation or a data acquisition device. Once the data is loaded into the environment, the user is presented with a representation of this data and provided with interaction methods to change the representation parameters interactively to obtain the best "view" required to gain more understanding. Each change of a parameter results in an update of the presentation. The data itself, however, does not change (see Figure 1.1).
Example areas where ISEEs are applied are plentiful. They can be found in medicine for the visualization of data obtained from medical scanners such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) [41, 166, 274]. Here a radiologist decides on the proper settings of a visualization environment for the visualization of a medical scan that aids a surgeon in assessing a diagnosis for a particular patient. In molecular biology, visual exploration environments are used to obtain a

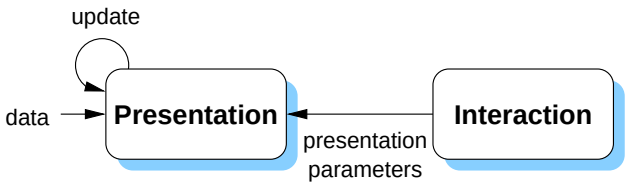


Figure 1.1: Schematic representation of an Interactive Static Exploration Environment (ISEE).

better understanding of the spatial structure of complex molecules [5, 92, 144]. In architecture, these environments can help in assessing the aesthetics of a building before it is built [150].

Data presentation and interaction

An important step towards a successful ISEE is to involve the researcher into the presentation as much as possible, thereby increasing the researcher's level of awareness [29]. To achieve this, an exploration system needs the following, mostly interdependent capabilities:

- Informative presentation – The most commonly used method to provide an insightful representation of complex data sets is by visualization. Here the abstract data are rendered as visual constructs that represent quantitative and relational aspects to the observer in an intuitive manner. This is the area of scientific visualization [27, 157, 170, 209] and information visualization [39, 222, 238]. Many visualization environments are available that provide means to efficiently achieve this [103, 106, 172, 242]. Although visualization is a very powerful and well understood method for data representation, other sensory modalities such as sound, touch and even smell, or combinations of different modalities may in some cases lead to better results.

- Persistence – The time that is needed to generate a rendering from start to finish can often not be easily dismissed. In the case of visual renderings, persistence (the rate at which consecutive frames are perceived as continuous motion) is obtained when at least 25 frames are rendered per second. Sound waves are perceived as tones from around 20 cycles per second and up. Humans can feel temporal frequencies starting from around 1 kHz and therefore rendering rates for haptic displays must be in the order of kilohertz. Update time is often dependent on the "level of detail" (LOD) in the presentation. In these cases a balance between the LOD in a presentation and the maximum allowed update time is necessary (the aim should be to employ "minimal means for maximum effect").

- Intuitive interaction – A prerequisite for effective exploration is that a sufficiently rich set of interaction methods is provided that allows a user to modify the parameters that control the presentation in order to be able to extract both

qualitative and quantitative information from the underlying data sets. An unfortunate side effect of increasingly richer sets of interactive methods is that user-friendliness is often compromised, so careful consideration is required during user-interface design. The ground rule for interaction capabilities is that interaction should be as intuitive as possible: intuitive interaction methods should require no explanation [171, 204].

- Rapid response – Some delay will always occur between the moment a user interacts with a presentation and the moment that the results are visible. This could be caused by many factors, including low tracking rates of input devices, communication delays and temporary reduced availability of computational or network resources. To attain accurate control over the environment and to avoid confusion with the user, the amount of lag should be minimized. In general, real-time interaction encourages exploration [29].

Provided these capabilities are carefully considered, these environments are well suited for the exploration of static multi dimensional data sets [13]. The design issues in the construction of an ISEE will be described in more detail in Chapter 2.

1.2.2 Dynamic environments

Interactive Dynamic Exploration Environments (IDEE) extend the previously described static model in that the information provided to the user is regenerated periodically. Again, the data originates from an external process which in this case is an active member of the exploration environment. The external process repeatedly generates new data, either autonomously (without user intervention), on-demand (as a result of user intervention) or both. In addition to the capabilities described for the static environment, the dynamic environment is expected to provide (1) a reliable and consistent representation of the results of the external process at that moment and (2) interaction mechanisms that enable the user to change parameters of the external process *and* of the presentation. Again, each change of a parameter in the external process or the presentation results in an update of the environment. New data generated by the external process results in an update of the presentation (see Figure 1.2).
While interaction in an ISEE influences the presentation only, in dynamic environments interaction influences both the presentation and the external process. On an implementational level this requires additional processing code to service interaction. On an operational level this change may influence the prerequisites described in this and the previous subsection. These design issues will be described in more detail in Chapter 5.
Interactive systems such as the ones described here allow for live experimentation by a researcher. These systems are called by many names; computational steering environments (CSE) [178, 245], user-steered calculation [100], problem solving environments (PSE) [84], human-in-the-loop (HITL) [235] computing, virtual laborato-

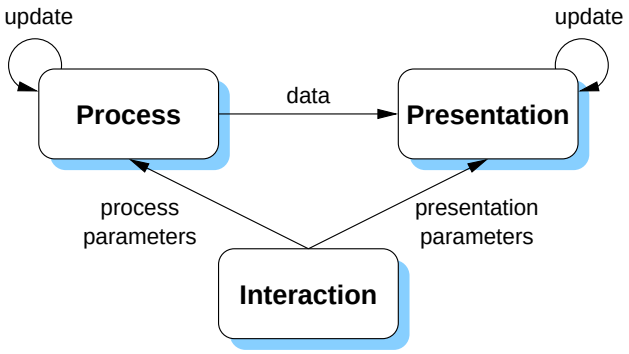


Figure 1.2: Schematic representation of an Interactive Dynamic Exploration Environment (IDEE).

ries [3] and even man-computer symbiosis [147]. In this thesis we will keep referring to "interactive exploration systems".

1.3 Scientific visualization

The key method to present data for the purpose of exploration in use today is visualization. The main reason for this is that the human visual system is thought to be the most important of the human senses; it has high bandwidth and allows natural communication as the human visual sensory system is capable of understanding complex shaped image renderings with relative ease. When visualization is applied to the representation of scientific data we speak of scientific visualization. A report of the National Science Foundation (NSF) in 1987 described scientific visualization as "the integration of computer graphics, image processing and vision, computer-aided design, signal processing, and user interface studies" [157]. As such, scientific visualization would not only entail the visual representation of scientific data but also image synthesis using computer graphics techniques, data analysis, engineering and human perception. This definition is too broad for our purposes. Here we will limit ourselves to a narrower interpretation in a scientific computing context, viz. the generation of visual representations from the results of scientific applications.
The basic objective of scientific visualization is to create a mapping of data structures to geometric primitives that can be rendered using computer graphics techniques (often points, lines, triangles, squares or polygons). Although this basic principle is easy enough, the real challenge in scientific visualization is to create a mapping that creates a representation that is understandable, correct and complete. Scientific visualization is applied in many different research and industrial areas. Often these applications involve the visualization of processes that are difficult to observe directly by the human visual and/or cognitive system. Phenomena of interest to the scientific and industrial community often include topics that can not, or not easily be studied directly because they are too small (molecules, atoms, etc.), too big (planetary systems, galaxies, etc.), too dangerous (nuclear explosions, combat situations), too slow

(colliding galaxies, stock rates), too quick (quantum processes), too concealed (organs in the human body), etc. Instead, methods are used to acquire and record data from the event of interest, the results of which are then visualized. In all cases it is important to realize that the visualization of these applications provide a *representation* of the underlying real world phenomena. This should always be kept in mind while interpreting the visualization as it could lead to misinterpretation.

1.3.1 Display techniques

In its earliest form, the visualization of scientific data was performed on systems that produced 2D vector representations. With the introduction of powerful raster displays and new algorithms, more complex graphics techniques came into reach of the scientist which included the calculation of 2D projected rasterizations of 3D scenes [79] and of lighting models such as smooth shading of polygonal surfaces [89, 183]. The dramatic increase in performance of graphics hardware over the last years allows these methods to be used for the rendering of complex 3D scenes in real-time. Thanks to the personal computer (PC) gaming industry, the cost of this graphics hardware has dropped dramatically as well, allowing scientists to create advanced scientific visualizations on cheap and commonly available hardware.

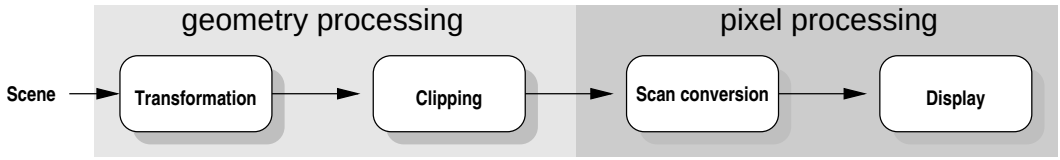


Figure 1.3: Simplified representation of a rendering pipeline.

A useful way to describe the method to create raster scanned images from 3D scenes is through a rendering pipeline [79]. A simplified form of a rendering pipeline is shown in Figure 1.3. The rendering pipeline is split into two parts: the *geometry processing* and the *pixel processing* phase. In the geometry processing phase, each primitive in the scene is traversed and transformed (translated, scaled, rotated) from model coordinates to world coordinates using a linear transformation matrix. Next, a viewing transformation is applied based on the location of the viewer and all primitives outside the view frustum are clipped. In the pixel processing phase, the clipped primitives are scanned into screen pixels (including lighting, shading, depth calculation and texture mapping). Finally, the resulting image is stored and composited (including alpha-blending and depth-buffering) and converted to an analog video signal by a digital-to-analog converter (DAC) for display on a monitor or projector.
Later and still ongoing research has resulted in ways to increase the performance of the graphics pipeline through multiprocessing and dedicated hardware techniques [4]. A major player in the success of computer generated 3D graphics in this respect was Silicon Graphics, Inc. (SGI), founded by Jim Clark in 1982 after having spent four years at Stanford University with the expressed purpose of developing hardware

technology that was called a "geometry engine". The geometry engine contained functionality that was able to almost instantaneously perform the complex geometrical mathematics required for 3D graphics that would otherwise have required thousands of lines of program code. It was not until 1985 that SGI put its first workstation on the market, but from that time on they continued to market systems with a graphical performance that was unsurpassed by other companies for many years to come. SGI's encounter with the Hollywood film industry in the early 1990s resulted in a huge exposure of Computer Generated Imagery (CGI), amongst which were blockbusters like *Terminator II* (1991), *Jurassic Park* (1993) and *Toy Story* (1995).

1.3.2 The visualization pipeline

The accepted approach to visualize data structures is through an extension of the graphics pipeline described in section 1.3.1. This "visualization pipeline" is, as the name suggests, a first-in-first-out (FIFO) structure where each stage accepts data at its input as soon as it is presented, transforms the data and provides its output to the next stage (see Figure 1.4). The "source" at the start of the pipeline reads or generates the data that is to be visualized. This data is transformed by one or more "filters" into a representation that is suitable for graphical rendering. A "mapper" transforms this representation into geometric primitives that can be drawn by the rendering engine. Finally, at the end of the pipeline the renderer takes the geometric primitives for transformation into pixelated images.

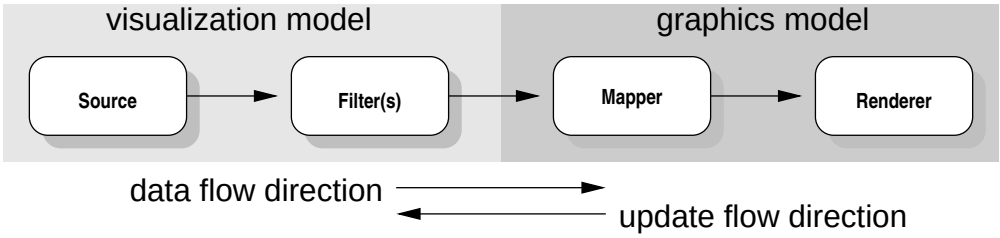


Figure 1.4: Simplified representation of a visualization pipeline.

Although some stages may be able to come up with reasonable defaults to perform their function, most stages in a visualization pipeline have to be configured in order to produce the results the user is looking for. To do so, each stage is configured using a set of parameters. A modification of parameters in a stage requires that at least this stage and all stages "downstream" are updated. To achieve this, the visualization environment checks the pipeline in the direction opposite to the data flow direction and instructs the first modified stage that it encounters to update itself. Once the modified stage is updated it produces new data, thus forcing subsequent stages to execute as well. In doing so, only part of the pipeline needs to execute which reduces computational overhead while maintaining consistency.
One example of a visualization environment that uses the pipeline architecture described here is the *Visualization Toolkit* (Vtk) [106, 209]. Vtk is an open source, freely

available collection of classes with contributions by a lively user group from all over the world. The types of functions in Vtk can be classified into graphics, image processing and visualization. The Vtk visualization pipeline is similar to that shown in 1.4 and will be described in more detail in section 4.2.

1.3.3 Interactive Scientific Visualization

The performance increase in computing and graphics hardware allow the long-term wish of many scientists to closely interact with their models. The NSF Scientific Visualization report envisioned a situation where large scale computations would be carried out on high performance computers while rendering and interaction would take place on a visualization workstation. The two would be connected by high-speed networks to cope with the high volumes of data that would have to be transfered. It would allow a researcher to connect to a running application, inspect its data structures and change values in order to understand its behaviour. This prospect would support a scenario that was considered not only possible but essential in the scientific discovery process.

With the increase in performance of computers, the possibility for researchers to perform larger and more complex simulations increases as well and, as a result, the demands posed on visualization workstations. To enable interactive exploration, the workstations would have to be able to do interactive rendering. At the same time, the complexity of the simulation models requires advanced visualization methods in order to represent the models and the simulation results in a comprehensive manner. This added functionality will reflect itself in additional complexity regarding the use of these environments, so improved user interfaces would be required to permit the operator to interact with the applications.

1.4 Virtual Reality

As noted in the previous sections, one way to support the scientific discovery process is by allowing the researcher to actively explore the processes that take place in his models. By increasing his involvement, the researcher would be able to gain a better understanding of the underlying models. A way to achieve this is to represent the data or processes in such a way that they can be “experienced” using the human senses. In general, as more of the human senses are stimulated by the events that take place around them, the more involved they get with the experience. The promise made by advocates of *Virtual Reality* is that computer technology facilitates the construction of devices that provide sensory stimuli to as many of our senses as possible in order to create the ultimate reality experience.

1.4.1 Historical background

There is no exact definition of what Virtual Reality (VR) precisely is. This is in part because of its historical association with a colourful diversity of social cultures, varying from game developers [141], arts movements [110], lyricists [11], visionaries that saw VR as the alternative drug [143] and other popularity writers [203]. Advocates have for some time even been reluctant to associate themselves with the term and have used alternatives such as "Artificial Reality" [137], "Cyberspace" [87] and later the most commonly used "Virtual Worlds" and "Virtual Environments". In this thesis we will use the name Virtual Reality (VR) for systems that are capable of generating interactive artificial worlds. We will use the term Virtual Environment (VE) for the environments created by VR devices.
Man has always looked for ways to escape reality and engage into other, more fantastic experiences. Some feel the origins of VR go way back to the time when primitive man painted pictures on the walls of caves [195]. Since the beginning of the written word, book and story-writers have recorded the figment of their imagination for the enjoyment by others. Playwrights left less to the imagination of their audience by enacting stories on stage with the sole purpose to provoke strong emotional responses. With the invention of the moving picture and television, cinematographers and television program makers captured the imagination of millions. Their stories became even more realistic over the years with the introduction of new technology that made the experience more compelling and breathtaking, such as the introduction of (surround) sound, larger screens, special effects and stereo projection techniques. However, when we compare them to the real world, all these alternative worlds have two major shortcomings before we may truly call them "reality experiences". First; we experience the world around us through our senses - our eyes, ears, nose, skin and tongue - yet, just a few of these are used to persuade us into believing we are "elsewhere". Second; few of them allow the viewer to influence the sequence of events that have been preprogrammed by the storytellers. They are all, essentially, non-interactive.

Virtual Reality pioneers

In 1956 Morton Heilig, a Hollywood based cinematographer, proposed that the next evolutionary phase after the cinema theatre would be an environment where the viewer experiences not only images and sounds but also odors and touch. Heilig believed that by doing so the barrier between the viewer and the theatre would dissolve, creating a total illusion which he called the "experience theatre". Heilig's work led to "Sensorama", a device he designed and patented in 1962 (see Figure 1.5). Sensorama was the first multisensory arcade game where the viewer would sit on a type of motorcycle. As the "passenger" was looking at three dimensional images of the Californian sand dunes through a binocular display, the motorcycle handhelds would tremble in sync with the images while breezes and odors were released from small grilles around the nose and ears. Although Sensorama was called a "game", the experience was prerecorded and played back for the user; the experience could not be controlled by the viewer and was therefore not interactive. Sensorama was no cash success but Heilig's

Figure 1.5: Morton Heilig's "Sensorama" (1962).

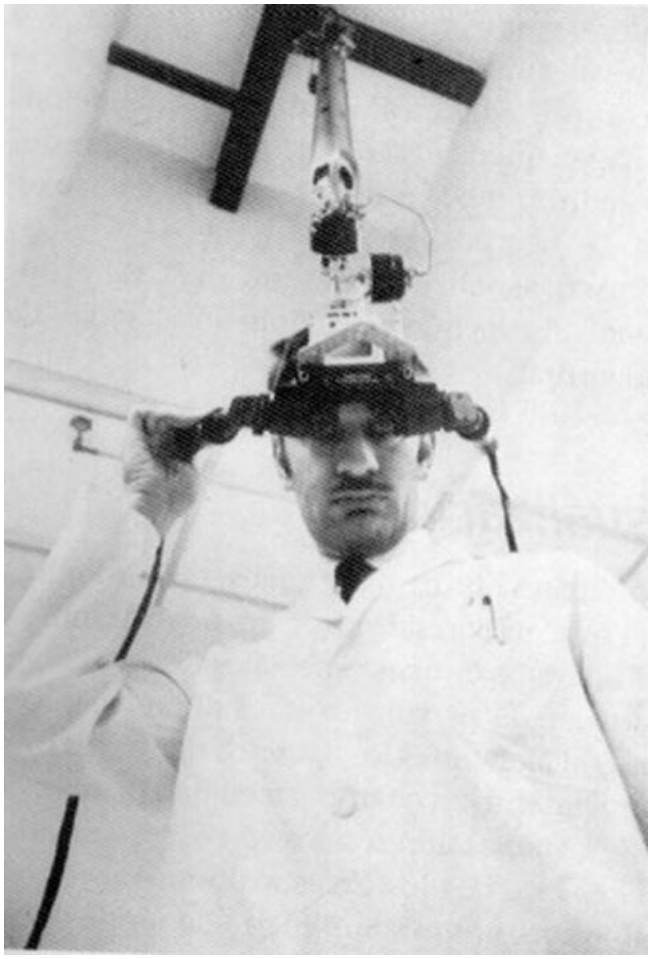


Figure 1.6: Ivan Sutherland's "Sword of Damocles" (1968).

use of multisensory stimulation testified of great vision nevertheless.

In 1965 Ivan Sutherland at the University of Utah proposed what he called the "Ultimate Display" [230]. This display would enable a person to experience a synthetic computer rendered "Virtual World" as if it were real. In 1968 Sutherland realized a binocular display which he called a "head-mounted display" (HMD) [231]. This device consisted of two cathode-ray tube (CRT) displays mounted on a helmet that projected images into the eyes giving the user a three-dimensional, stereoscopic view of the generated images. The helmet was connected to a contraption, aptly named "Sword of Damocles", which was suspended from the ceiling and could track the position of the wearer's head (see Figure 1.6). When the user moved his head, a computer would recompute the images rendered on the displays so that the user would get the impression that the virtual objects were stationary as the user moved around. Sutherland did not pursue the development of wearable displays because the technology available to him at that time was too primitive. Instead, he turned to work on the fundamentals of computer graphics hardware and software design. In 1968 Ivan Sutherland

together with David Evans founded "Evans & Sutherland" (E&S), a company that received high credits in the development of advanced graphics systems that were used in aviation and military simulators. Sutherland received the 1988 *ACM Turing Award* for his numerous contributions to computer graphics and the 1998 *IEEE John von Neumann Medal* for "pioneering contributions to computer graphics and microelectronic design and leadership in the support of computer science and engineering research".
The work by Sutherland inspired many scientists in different research areas. One of them was Frederick Brooks, Jr. who in 1971 used scientific visualization techniques and a force feedback device for the representation of large molecules [28]. Around that time, VR technology had also slowly progressed to a state where large industries were seeing its potential [150]. During the Apollo missions of the late sixties, NASA used simulators that were used to simulate docking procedures of the Lunar Excursion Module (LEM) and the command module (CM). Another application of VR technology emerged in the aviation industry in the form of flight simulators. Flight simulators are used to train pilots of aircrafts before embarking on their first actual flight. These devices would use graphic displays, sound systems and motion platforms that could realistically reproduce the behaviour of an aircraft during all stages of air travel, from take-off to landing, from normal situations to the most dangerous scenarios. The defense industry used simulators to train soldiers before missions in "war game" systems that were connected to a distributed virtual environment called SIMNET, linked in real time, involving armored tank simulations [154, 220]. These combat simulations systems were used in the preparation of "Operation Desert Storm". Other research areas where VR technology is deemed to have great potential is in training and education [158].

1.4.2 A taxonomy of VR systems

Exactly what may be called a VR system remains a topic of debate. Some feel that a VR system should be able to generate an environment that is indistinguishable from the real world. Some even go the lengths of developing an extension of the Turing test to this end [10]. Others say that a VR system should be able to immerse the viewer into an artificial, yet convincingly real environment. These systems would have to obscure the real world and at least have to track the head's position and orientation. Still others say that a VR system must track the user's head but may use a desktop display. From a marketing standpoint, just about any system that displays animated 3D graphics seems to merit the VR label. The following types of VR systems can be distinguished (based on a taxonomy by Jerry Isdale [108]):

- Desktop VR - These systems are also sometimes called "Window on World" systems (WoW). In these systems, the conventional desktop monitor is considered as a window onto the virtual world. More realism can be obtained through stereoscopic images, often produced through liquid crystal display (LCD) shutter glasses, which make images "pop out" of the screen. The term "Fish Tank VR" is used when these systems are augmented with a head tracker [250]. The head

tracker is used to change the view on the virtual world based on the position of the wearer. The resulting effect, called "motion parallax", gives the viewer powerful clues as to the relative distance and size of the objects in the virtual world.

- Video Mapping - In these systems a video input signal of the user in the real world is mixed with the virtual world. The user watches a monitor or a projection on a large screen that displays the silhouette of his body interacting with the virtual world. The most famous example of this type of VR was the "Mandala" system [150]. Because the viewer looks at himself interacting with the virtual world on the display, this type of VR is also called "world centered" or "second person" VR.

- Immersive Systems - Immersive systems immerse the viewpoint of the user inside the virtual world. To accomplish this they often use display technology that engulf the viewer as much as possible, such as HMD devices or multiple large projection screens, in an effort to overwhelm the viewer with a view on the virtual world and distract him from the real world. HMDs in this respect are sometimes said to provide an "out of body experience" since the user will not be able to see his own physical appearance, or any other physical object in the real world for that matter. As the viewer in an immersive VR system experiences the virtual world almost directly, this type of VR is also called "user centered" or "first person" VR.

- Telepresence - In telepresence, the virtual world is not necessarily artificial. Instead, remote sensors, such as video cameras, are used to act as "extension cords", linked to the human senses to create a on-location view of a real world experience. Examples of where these systems are in use is with firefighters and bomb squads where remote controlled robots are equipped with cameras and robotic actuators to assist during hazardous situations.

- Augmented Reality (AR) - In these systems the user's view of the real world is enhanced or augmented with additional information generated from a computer model. This type of VR is also called "Mixed Reality" or "see-through VR". Display devices that are used in these systems are called heads-up displays (HUD); they often consist of semi-transparent material that allows the viewer to watch the real world while computer generated images are overlaid providing additional information. Examples are systems used by fighter pilots where the displays provide cockpit information on the inside of the pilot's helmet.

In this work we focus our discussions on the use of immersive VR systems, such as the SARA CAVE that will be described in section 1.5, although we will in some cases address the use of desktop systems as well, such as the UvA-DRIVE system described in Chapter 3 [14].

1.4.3 Components in a VR system

Any interactive VR system consist of three components [265]. The first is the rendering component which transforms abstract data into a representation that can be perceived by one or more of the human senses. The display component converts these renderings into sensory stimulations for the human perceptory senses. The last is the interaction component which is responsible for determining the position and gestures of the viewer to which the environment should react.

3D computer graphics

Most of today's high performance graphics workstations contain dedicated hardware components that accelerate the various stages of graphics rendering in the pipeline such as those depicted in Figure 1.3. The hardware architectures vary from vendor to vendor but most use dedicated memories (for textures, depth buffers and frame buffers) and processing units (for transformation and lighting, shading, antialiasing, z-buffering, image composition and texture mapping) that interact in parallel. These dedicated graphics systems relieve the main central processing units (CPU) for doing other tasks. One of the main vendors of computer graphics chips today, nVidia, has coined the term Graphics Processor Unit (GPU) for these devices.
The industry standard Application Programming Interface (API) to take advantage of these graphics devices is OpenGL [16, 189, 261]. The OpenGL standard was specified in order to encapsulate device-dependent code and thus to promote program portability to other operating systems. The interaction between the application and OpenGL takes place on the level of polygons, light sources and linear transformations to specify 3D scenes. On a higher level, frameworks such as OpenInventor, World Toolkit and Performer allow the user to specify high-level 3D scenes using object-oriented methods [190, 212, 236]. These frameworks encapsulate many of the more intricate complexities associated with increasing rendering performance (such as scene culling) and interaction with 3D scenes (such as object intersection).

Stereoscopic display technology

VR display technology can be subdivided in two categories: head mounted displays (HMD) and head tracked displays (HTD). The difference between the two is that with HMD systems the display is connected to the viewer's head, for example in the form of a helmet, so that wherever the viewer's head moves, the display moves. In HTD systems the display is stationary, as with desktop monitors or projection screens.
In both cases the display must portray a stereoscopic pair of images to the viewer so that he sees the virtual objects as if they float in front or behind the display. The common method used to achieve this is to generate stereo image pairs, one for the left eye and one for the right eye, that are displaced in the same way as when a real object is floating in front of the viewer. In stereoscopic HMD systems, these stereo pairs are displayed on two separate displays, using either cathode-ray tubes (CRT) or liquid crystal displays (LCD), mounted in front of each eye. Based on the position and

orientation of the viewer's head, the left and right projections of the VE are calculated and rendered onto their respective displays.

In HTD systems the stereo image pairs are rendered onto one and the same display. Here the viewer wears a device that separates the left and right eye images into the correct eye. This separation must be performed accurately. If not, images intended for one eye may "bleed through" to the other which makes it difficult for the human brain to fuse the image pairs into one stereoscopic image. The most common methods to separate image pairs into the correct eye are through bichromatic image pairs, time sequential frames or image polarisation. In bichromatic stereo, the left and right eye views are each rendered in a different colour (often blue/red or green/red) while the viewer wears glasses with the same coloured filters in front of each eye. The filters absorb light of the same colour while passing the other. A major disadvantage of this technique is that all colour information in the images is lost. Time sequential systems render left and right eye views in rapid succession after each other in time while "shutter glasses" block the image for one eye and allow it to pass for the other. As long as the frequency at which this takes place is higher than 50 cycles per second, the viewer will see a flicker free stereoscopic image. Yet another technique is to use filters that change the polarization direction of the light emitted by the display differently for each image. The most common system use linear polarization filters placed in an orthogonal configuration (for example; up/down for the left eye and left/right for the right eye). In this case, the viewer wears polarized filters that only pass light that is polarized in the same direction as the filter. An advantage of this technique is that these (passive) polarized glasses are relatively cheap compared to (active) shutter glasses and that the images are flicker free. A disadvantage is that the viewer should always keep his head in line with the polarization direction of the emitted light. If the viewer tilts his head, images intended for one eye will bleed through to the other, destroying the stereo effect. Circular polarization filters do not have this problem, but unfortunately these filters are more expensive than linear polarization filters while projection screens that do not distort circular polarization are even costlier. Note that the techniques described here all block half of the light emitted by the displays. As a result, the viewer always perceives an image that is half as bright compared to the original.

The decision on which type of display is "best" for a specific application depends mainly on the indirect consequences of the categories. HMD devices isolate the viewer from the real world, in particular the types that are completely closed. The viewer will not be able to see his office, his colleagues, or even his own hands. In HTD systems, the viewer is still able to see objects in the physical world which make them more suitable in collaborative applications. If more than one person should be able to take part in the virtual experience, a HMD based setup requires that each viewer wears a display, possibly with its own graphics system. In HTD systems more than one person can share the same display space and make use of relatively cheap glasses. HMD systems require less space than HTD systems but HMD system can also be quite heavy to wear. Also, HMD systems suffer more from the effect of delays in the system than HTD systems. As there is always some delay between the moment the viewer moves

his head and the moment the newly calculated image appears, HMD systems often suffer from "lag". This lag is caused by delays in obtaining information on the head's new position and orientation and the recalculation of the new images. Depending on the length of these delays, the viewer can feel disorientated or even nauseated, a phenomenon that resembles see-sickness but in the context of VR research it is sometimes referred to as "cyber sickness".
A recent development in VR display technology are autostereoscopic displays. These devices are based on LCD displays, overlayed with an optical prism sheet that is aligned in such a way so that a given column of pixels is only seen by one eye, and not the other. The obvious benefit of these displays is that the viewer does not need to wear any eye-wear. However, as a result of the technique, the effective resolution is greatly diminished. Also, the displays often require the viewer to remain within the frustum in which the stereoscopic effect can be seen.

VR interaction technology

A vital ingredient in the development of *interactive* VR are the input devices that sense position, posture and gestures of the user. Tracking sensors form an integral part of many VR systems developed over the years and are most often used to track the position and orientation of the user's head and hand at the least. An important step in the development of unencumbering devices that could track the position and orientation of a physical object was taken in 1979 by Raab et al. with the development of an electromagnetic sensor known as the "Polhemus" [192]. These sensors are capable of measuring position and orientation at six degrees of freedom (DOF), three translational and three rotational, with reasonable accuracy.
Electromagnetic tracking systems use a transmitter that contains three orthogonal coils that are energized sequentially first for the x axis, then y, then z. The currents through the coils generate a magnetic field that generates a current in the receiver which also contains three orthogonal coils. When each transmitter coil is energized, the current through all three receiver coils is measured. This results in 3 receiver values for each of the 3 transmitter coils. The result is a system of linear equations that have enough unknowns to resolve the position as well as the orientation of the receiver relative to the transmitter.
Today, there are still mainly two companies that produce magnetic tracker products: Polhemus and Ascension, a company founded by Ernie Blood who was a former employee of Polhemus (both companies are located in Vermont) [253, 257]. The main difference between the two types is their sensitivity to the vicinity of metal objects. The trackers developed by Ascension use a direct current (DC) to energize the transmitter coils rather than an alternating current (AC) as used in the Polhemus trackers. A DC current will, after an initial spike, generate no current in any surrounding metal instead of a constantly changing current with AC systems. Also, the Ascension devices turn the transmitter off between measurement cycles to take an additional set of readings to compensate for ambient electromagnetic fields. This combination makes the Ascension devices more accurate in environments with metal

objects. Tracking devices developed by other companies like InterSense [255] use different techniques, such as ultrasound, inertia, images obtained from cameras or hybrid techniques that use combinations of these. Critical specifications of tracking sensors are [128, 129, 160, 192] (see also Table 1.1):

- Resolution: the smallest displacement that can be measured by a sensor. In tracking systems where the position and orientation of a sensor is determined relative to a source, the resolution frequently depends on the distance of the sensor to the source.

- Accuracy: the deviation from the reported position and orientation of a sensor to its actual position and orientation. In tracking systems where the position and orientation of a sensor are determined relative to a source, the accuracy frequently depends on the distance of the sensor to the source.

- Update frequency: the number of updates the sensor is able to provide per second. In some cases the update frequency decreases as more sensors are used simultaneously in the same setup.

- Latency: the time lapse between a displacement of the sensor and the moment this is available at its output. The latency often depends on the type of interface between the tracking and the computing system (i.e. serial connection, ethernet, etc.).

- Jitter: the amount of reported displacement changes when the sensor is held still.

- Range: the volume in which the sensor can be used accurately. Note that some systems use technology that requires a line-of-sight between the source and the sensor.

- Drift: the accumulating error, over time, of the reported position and orientation of a sensor to its actual position and orientation. To compensate, sensors must either be recalibrated after a longer period of use or, otherwise, be augmented by other types of sensors.

1.5 The Cave Automated Virtual Environment

The Cave Automated Virtual Environment (CAVE) developed by DeFanti et al. at the Electronic Visualization Laboratory (EVL, University of Illinois at Chicago) is a fully-immersive, projection based VR device that was first presented at the ACM SIGGRAPH conference in 1992 [52, 53]. The CAVE installed at SARA in 1997 by Silicon Graphics and Pyramid Systems consists of the following components [205].

	Magnetic	**Ultrasound**	**Optical**	**Inertial**	**Mechanical**	**GPS**
DOF	6	6	3/6	3 (orientation)	6	3 (location)
Resolution: position	0.5 mm at 30.5 cm ±50 cm at 3 m	0.5 cm	0.2 - 7 mm	N.A.	0.5 mm	5 m
Resolution: orientation	0.1° at 30.5 cm ±17° at 3 m	5°	0.01°	0.02°	0.1°	N.A.
Accuracy: position	1.8 mm	5 cm	0.4 mm	N.A.	1 mm	2 m
Accuracy: orientation	0.5°	5°	0.02°	3°	0.5°	N.A.
Update frequency	120 Hz	20-50 Hz	60 - 600 Hz	180-500 Hz	70 Hz	1 Hz
Latency	4-20ms	60 ms	1 - 60 ms	2 ms	1 ms	2 s
Jitter	low	high	low-medium	low	low-medium	medium-high
Range	3 m	10 m	0.5 - 3 m	360° all axes	1-2 m	outdoors
Drift	medium	medium	low	high	low-medium	low
Cost	$6000	$500	$1k - $50k	$1000	$1500 - $95k	$500

Table 1.1: Typical specifications of different types of tracking systems [1, 128, 129, 160, 192, 253–257].

1.5.1 Computing and graphics hardware

The stereoscopic images in the CAVE are generated by an SGI Onyx2 RealityMonster [217]. The machine at SARA was one of the first delivered by SGI and, at the time of installation in 1997, the InfiniteReality hardware was among the most powerful general purpose graphics systems available. SARA's CAVE configuration consists of the following computing and graphics components:

- 8 MIPS R10000 processors, each running at 195 MHz clock speed, each with 4 MB second level cache,
- 8×128 MB memory configured into 1 GB main memory using a cache-coherent Non Uniform Memory Access (ccNUMA) architecture,
- 4 InfiniteReality2 graphics pipes, each consisting of 4 Geometry Engines, 2 Raster Managers with 16 MB texture memory and a Display Generator,
- over 100 GB home file system on a dedicated O200 fileserver,
- HIPPI High speed network interface (800 Mbit/s),
- 8 channel digital audio system (ADAT). The CAVE uses four audio channels which allows front/back and left/right positioning of sounds.

The hardware architecture of the InfiniteReality2 graphics interfaces is configured into a pipeline. Each pipe can be subdivided into multiple channels (up to a maximum) where each channel is handled sequentially within one pipe. The performance of a channel is therefore roughly the performance of one pipe divided by the number of channels. The "Geometry Engines" provide the main interface to the main CPUs. They perform object transformation (scaling, translation, rotation), subdivide polygons into triangles and perform the projection of 3D world coordinates to 2D screen

coordinates. The “Raster Managers” convert the projected triangles into pixel representation using a scan conversion technique. Finally, the “Display Generators” convert the digital pixel streams into an analog video signal that can be displayed by a monitor or projector.

1.5.2 Projection system

In order to fit the SARA CAVE on the available floorspace, foil mirrors are used to reflect the images onto the CAVE’s projection screens (see Figure 1.7). The projectors used in the SARA CAVE are Electrohome Marquee 8500 which are capable of handling resolutions up to a maximum of 1280 by 1024 pixels at 120 Hz. The CAVE uses an active system to produce stereo images. The graphics pipelines alternately generate the images for the left and right eye while the Stereographics CrystalEyes [49] liquid crystal shutter glasses that the viewers wear open and close synchronized with the images on the screen. This synchronization signal comes from a number of infrared emitters placed around the CAVE.

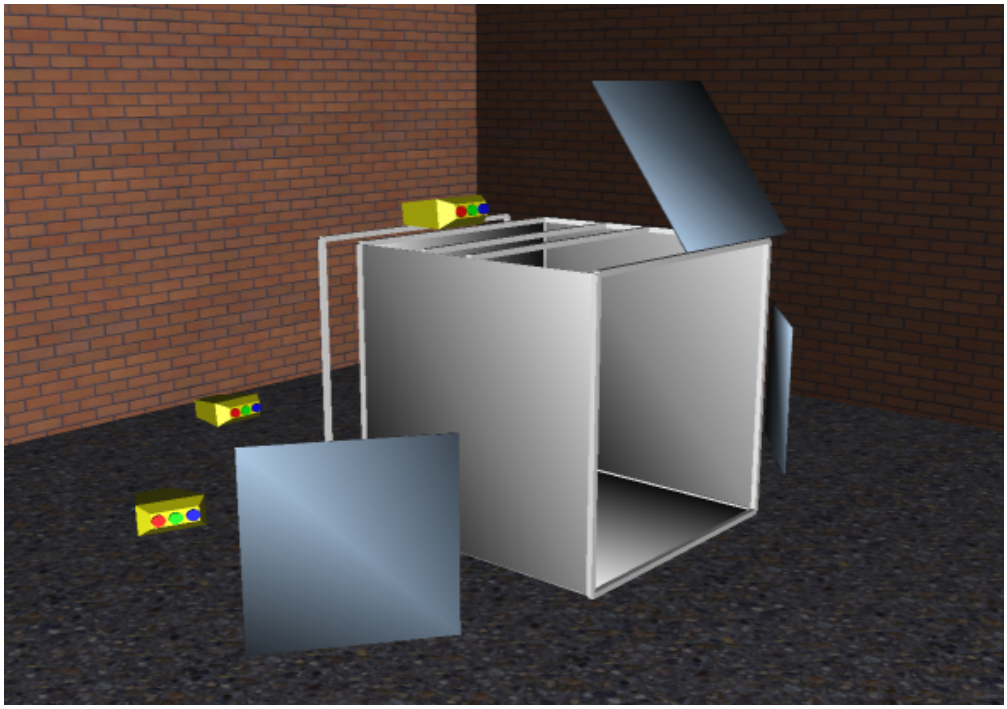


Figure 1.7: Projectors, mirrors and screens setup of the CAVE used at SARA.

Projection based VR systems have several advantages over “classical” HMD systems. First; projection based systems allow more than one person to share the same VE at minimal cost. In practice there are practical limits to the maximum number of people, which are often caused by the physical dimensions and placement of the projection screens. For example, although the floor area in the CAVE is 9 m^2, more than 5 people in the CAVE at the same time will quickly get in each other’s way. Of course,

when multiple HMDs are properly connected together they could also be used to allow multiple persons to share the same environment, but the cost in additional hardware would be far greater than that of the additional shutter glasses in a projection system. Moreover, projection based systems do not obscure a human's peripheral vision as HMD systems do. The user will therefore be able to see physical objects as well, most importantly his own limbs and fellow researchers, but also objects that can be useful while exploring the VE. This feature, however, also points us at a hindrance of projection based systems; virtual objects will be obscured from view by real objects if these are in the line of sight from the viewer to the projection of the virtual object on screen, even if the real object was really behind the virtual object. This can make interaction with VEs cumbersome in some situations. Also; the images in the CAVE are drawn in correct perspective *only* from the tracked user's viewpoint. People next to this user see the same perspective, *as if* they were standing at the exact same location (see Figure 1.8). Since they are not, a user pointing his finger at a virtual object will seem to point at something completely different from a guest's viewpoint [200]. To resolve both of these issues, a *virtual* pointer is often used, represented in the same virtual space. It therefore suffers from the same change in perspective as all other virtual objects. The stereoscopic representation of the virtual pointer, together with other depth cues such as partial occlusion caused by the intersection of the virtual pointer with the objects under investigation provide a sufficient solution to these problems.

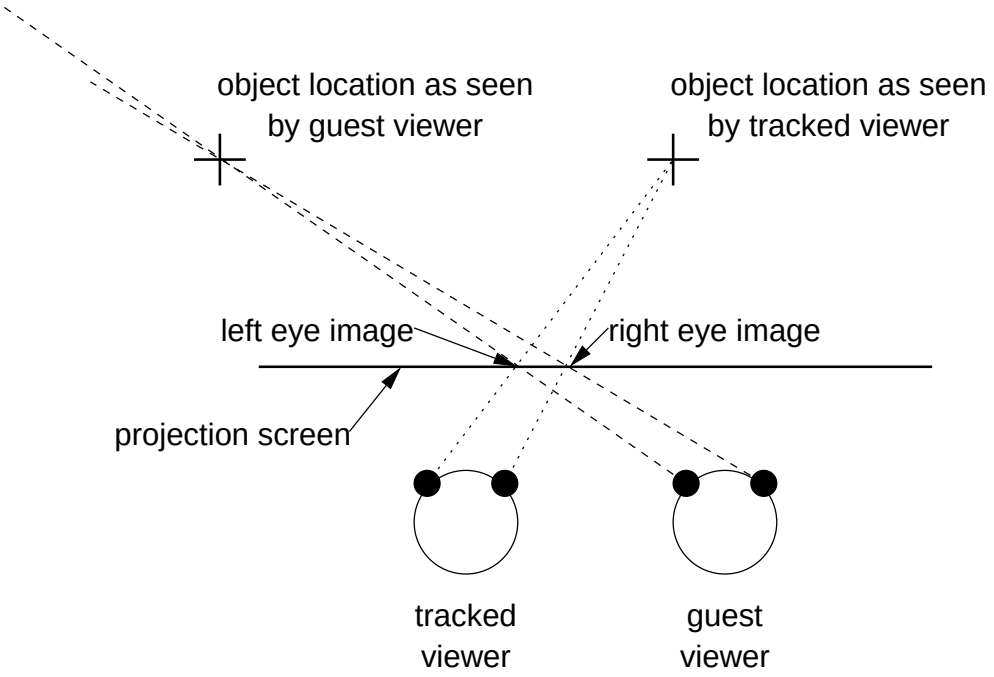


Figure 1.8: Difference in perceived location of a virtual object as seen by the tracked viewer and a guest viewer.

1.5.3 Interaction devices

To be able to generate the correct perspective view, the computer needs the position and orientation of the user's eyes. To measure this the CAVE uses an Ascension Technologies Flock of Birds (Extended Range) electromagnetic tracking system [45], with one of the sensors attached to the user's shutter glasses. A second sensor is attached to a "wand", a hand-held device with 3 buttons and a small joystick that is used to navigate through and interact with the virtual environment. Additional tracking sensors are available for application specific purposes.
Other input devices that are in use (or have been used) in the SARA CAVE include a 15 sensor Ascension MotionStar Wireless full-body motion capture setup [46], a pair of CyberGloves [48], PC-joystick interfaces (joysticks, gamepads, etc.) that can be used via an Unwinder box [184] and a wireless microphone that can be used for audio-conferencing or in conjunction with SARA's speech recognition software (CAVETalk) which is based on IBM's ViaVoice package [47].

1.5.4 Writing CAVE applications

The software environment of choice for the development of CAVE applications is the "CAVE library" (CAVELib) [249]. CAVELib provides an application programming interface (API) that abstracts from the VR hardware and allows VR applications to run on systems ranging from HMDs to multi-walled projection based displays with little to no changes to the program. More information on the structure of CAVELib applications and an analysis of some of its design problems are provided in Appendix A. Other APIs that support CAVE-like environments include the WorldToolkit [212], Bamboo [251] and Avocado [237]. Open source and research initiatives (some of which include support for CAVE environments) include VRJuggler [241], FreeVR [216], DIVE [40], MAVERIK [101], PVR [244], and VIRPI [86].

1.6 Overview of this work

This thesis describes the technological, application and scientific issues involved in the design of environments for the purpose of interactive exploration in virtual environments.
Chapter 2 begins with an analysis of the situations in which virtual environments can provide significant benefits over traditional exploratory data analysis and describes the issues involved in the design and implementation of interactive exploration environments for use in virtual environments. These issues are illustrated by the description of a number of test cases.
Chapter 3 describes the design and construction of a low-cost virtual reality system, the *Universiteit van Amsterdam Distributed Real-time Interactive Virtual Environment* (UvA-DRIVE), built out of inexpensive, off-the-shelf hardware, and shows how this system can be used for interactive exploration at a fraction of the cost of similar commercially available systems.

Chapter 4 describes interaction methods for use in virtual environments to support scientific exploration. Interaction is at the center of any type of exploration. We describe several techniques that facilitate the construction of information-rich and highly interactive virtual environments.
Chapter 5 enhances the static design concepts towards the design of Interactive Dynamic Exploration Environments (IDEE). We will see that the design of these systems benefits from a distributed architecture where the various components execute on different systems and describe how these systems can be used for human-in-the-loop experimentation.
Chapter 6 describes an IDEE which was built as a test case to validate the ideas described in the previous chapters. This environment combines distributed simulation, interactive dynamic exploration and virtual reality technology into an interactive simulated vascular reconstruction operating theatre that allows pre-operative studies of abdominal vascular reconstruction procedures.
In Chapter 7 we recapitulate all issues addressed in this thesis and reflect on the design decisions that have been taken in the progress of this work. Directions for future research are described as well as recent scientific and technological developments that are relevant to this work.

Chapter 2

Design considerations for interactive exploration environments*

"Good design comes from experience, experience comes from bad design."

Frederick P. Brooks, Jr.

2.1 Introduction

In this chapter we investigate the different issues that are involved in the development of an exploration environment that allows interactive exploration of large data and parameter spaces. We distinguish three different aspects in such a system: the available technology, the applications and the scientific issues.

2.1.1 Technological issues

The technological aspects are mainly concerned with the computational hardware that is used for the exploration system. The increase in capabilities of modern computer systems has been impressive, in some cases even allowing both the computation and presentation tasks to be executed on the same machine. However, a performance increase may be attained by running these tasks on dedicated machinery. For example, many simulation applications perform better on dedicated hardware such as vector processors, massively parallel platforms or other high performance computing systems. Also, state-of-the-art graphical systems are now available that are well suited for the presentation tasks. When the simulation and presentation tasks are

*Parts of this chapter have been published in *R.G. Belleman, J.A. Kaandorp and P.M.A. Sloot. "A virtual environment for the exploration of diffusion and flow phenomena in complex geometries",* Future Generation Computer Systems, 14(3-4):209–214, 1998.

distributed over different systems, some means of communication is required between the two.
An important goal towards a successful exploration environment is to involve the researcher in the presentation as much as possible, thereby increasing the researcher's level of awareness [29]. To achieve this, an exploration system needs the following, often conflicting capabilities:

- Quality of presentation – The most common method to provide insight in simulated phenomena is to represent the abstract data as visual geometric constructs that present quantitative and relational aspects to the observer in an intuitive manner. Many scientific visualization techniques are now available that provide means of efficiently achieving this [209, 242]. However, for some applications mere visual exploration will not always be sufficient. These applications can benefit from the integration of additional sensory modalities to increase the user's field of perception. Additional modalities applied in previous research include the use of sound [68, 93, 141], touch to provide "haptic feedback" or "tactile feedback" [28, 33, 36, 65, 66, 91, 109, 153, 224] and even smell [125, 138].

- Rapid frame rate – While the capabilities of modern graphical workstations allow the construction of high quality and complex images with relative ease, the level of detail in the presentation should be minimized to avoid information clutter and to achieve high frame rates (the aim should be to employ "minimal means for maximum effect"). For a usable exploration environment the visual frame rate should be at least 10 frames per second. Insufficiently high frame rates can lead to disorientation and even nausea, a phenomenon which is often called "cybersickness" in the context of virtual environments and is akin to motion sickness [131, 156]. Note the difference between "frame rate" and a display's "refresh rate" which is the frequency at which a display device redraws an image on screen. It is generally accepted that a display refresh rate of more than 60 Hz results in a stable, flicker free image. Also note that a frame rate that is higher than a display's refresh rate results in wasted computing power.

- Intuitive interaction – Some level of interaction with a presentation is mandatory. Unlike the standard interaction metaphors used in windowing systems on workstations, no standard user interaction metaphors yet exist for virtual environments. Most notably in the "living simulation model", the increase in functionality expected from an exploration environment demands a well considered user interface.

- Real-time feedback – Some delay will always occur between the moment a user interacts with a presentation and the moment that a response is available. This is caused by low tracking rates of input devices, communication delay between the exploration and the simulation system and temporary reduced availability of computational or network resources. To attain accurate control over a running application, the total amount of lag should not exceed a couple of seconds. If

longer, users tend to think that their interaction was not recognized and should be repeated or that the system is defective [171].

2.1.2 Application issues

Each application has different execution characteristics with respect to the update frequency, the point in the application where interaction may occur and at what moment, and the amount of data that changes in between each update and thus needs to be communicated to the presentation system.
The relative importance of each of these capabilities depends primarily on the characteristics of the simulation application and the limits imposed by the available infrastructure. It is our intention to develop a conceptual model with which we can describe the behaviour of an exploration system. To obtain an accurate representation of the behaviour of the system as a whole, this model should include the temporal characteristics of the application, the presentation software, the interconnection between the two, and the hardware each runs on.

2.1.3 Scientific issues

The key issue towards a successful exploration environment is real-time interaction [29]. One way to achieve this is through a trade-off between speed and accuracy of the exploration by allowing "short-cuts" to be made in a guided search by an interactive feedback mechanism between the application and the exploration system. The scientific question we want to address is concerned with the implications of making these "short cuts".
The guided search algorithms that are used to search a problem's parameter space are often based on methods that stem from mathematical, biological or physical models at a macroscopic level. To be able to perform computer simulations, these macroscopic models must first be converted into a computable algorithm. At this microscopic level, it is expected that the computable algorithm provides a sufficiently accurate abstraction from the macroscopic model such that it achieves the desired result. However, the process of making "short cuts" on a microscopic level may have severe implications on the model that is being simulated. Indeed, the resulting simulation may well be totally different from that described at the macroscopic level.

2.1.4 Test cases

As an initial assessment of the available technology and to obtain a better understanding of the application and scientific issues involved in interactive exploration systems, we built a number of static environments for applications from different research areas.

2.2 Car crash simulation playback

The PAM-CRASH package developed by the ESI Group is a numerical simulation package for prototyping and manufacturing processes that take into account the temporal mechanics of car crashes at high accuracy [75]. The ESI Group's product portfolio provides a virtual engineering solution known as the "Virtual Try-Out Space" (VTOS). This solution allows reduced costs and development lead times by progressively eliminating the need for physical prototypes. The PAM-CRASH software is based on a Finite Element simulation model. Depending on the desired accuracy, simulations can take days to complete on even the most powerful computer. In the ESPRIT "CAMAS" project the Section Computational Science of the University of Amsterdam investigated typical data dependencies in order to parallelize the PAM-CRASH code [58]. PAM-CRASH was also used as a test-bed in the Esprit project "Dynamite" where it was investigated how to use computing cycles on idle workstations in an organization through dynamic load balancing [44, 243].
When a simulation run is complete, the data is visualized on a desktop computer system for inspection. The ESI Group was interested in knowing whether a VR device like the SARA CAVE would give additional benefits to the visualization of simulation results compared to desktop methods.

2.2.1 Implementation

A number of PAM-CRASH simulation runs were performed by the ESI Group and the resulting data files were gathered for visualization in the SARA CAVE. Each data set consists of multiple files, one for each time step, with the number of time steps varying between tens to close to one hundred. Each file contains a description of the simulated object stored as quadrilateral patches consisting of four vertices and a normal vector for each vertex. We designed and implemented a program that allows the simulation results to be played back in a virtual environment (see also Figure 2.1). The program is implemented in C [126], uses CAVELib [249] for control over the VR equipment (in this case the displays and interaction devices in the SARA CAVE) and OpenGL for graphics rendering [16, 189, 261]. In this case, OpenGL uses the vertex and normal information to render the object under simulation as a smooth shaded surface. To allow for interactive playback of the simulation results, all time steps are read in at program startup and compiled into OpenGL "display lists" [261]. A display list consists of one or more OpenGL commands that are compiled into a format that can be executed more efficiently by the OpenGL rendering engine. In general, the time required to execute a display list is significantly lower than the sum over the execution time of its constituents. Therefore, the use of display lists increases frame rate which in turn increases responsiveness of the system as a whole. This design choice means that some time is spent between program startup and the moment the user can start using the program but since this time never exceeds approximately one minute, even for the largest data sets, this was considered acceptable.
After initialization, the VE represents the first time step of the simulation. In ad-

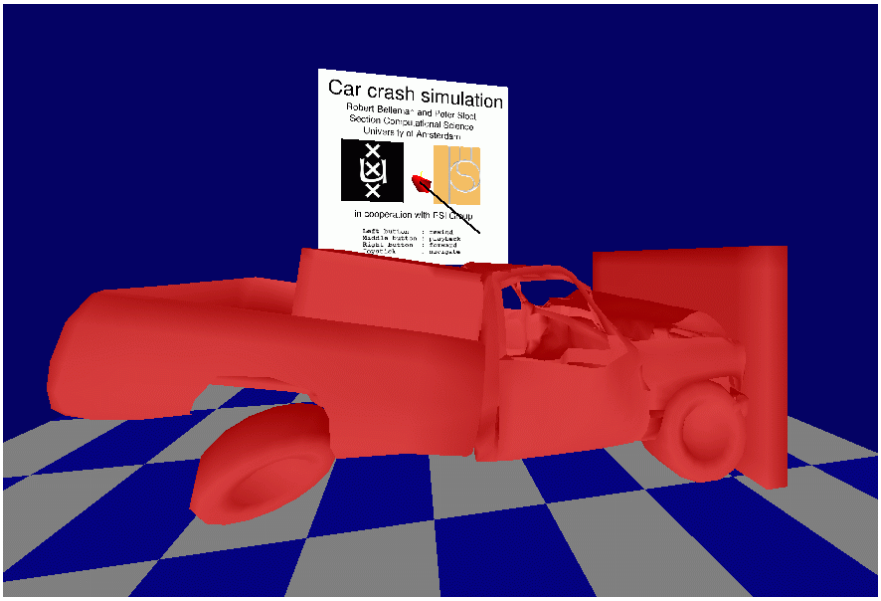


Figure 2.1: Car crash simulation playback in the CAVE (see also colour reproduction on the back cover).

dition, a virtual pointing stick is drawn from the front of the wand, represented by a simple line, which can be used by the user to indicate points of interest to fellow viewers. This simple mechanism turns out to be an adequate solution to resolve the problem that exists in most stereoscopic projection based VR systems, namely that the perceived location of a virtual object as seen by the tracked viewer differs from that perceived by a guest viewer (as previously described in section 1.5.2 on page 19). Using the three buttons on the CAVE wand, the user plays back the crash simulation much in the same way as the "fast forward" and "rewind" buttons on a video cassette recorder (VCR); while pressed, the right button plays the simulation forward in time, the left button backwards in time. Pressing the middle button replays the simulation from start to finish. The wand's joystick is used to "navigate" through the VE. Navigation provides users with methods to move beyond the confinements of the CAVE's physical dimensions. Objects beyond the CAVE walls come into reach by moving the CAVE towards them. Note how this concept places the user of the VE in the center of this type of interaction; the user is transported from one place to another while the objects remain where they are. As shown in Figure 2.2, pressing the joystick forward moves the user towards the direction in which the wand is pointing, pulling the joystick backwards moves the user away from the direction in which the wand is pointing. Pressing the joystick sideways rotates the view left or right.

2.2.2 Experiences

Although the information in the data sets only contains a description of the deforming structure over time, the program can easily be extended to visualize any additional

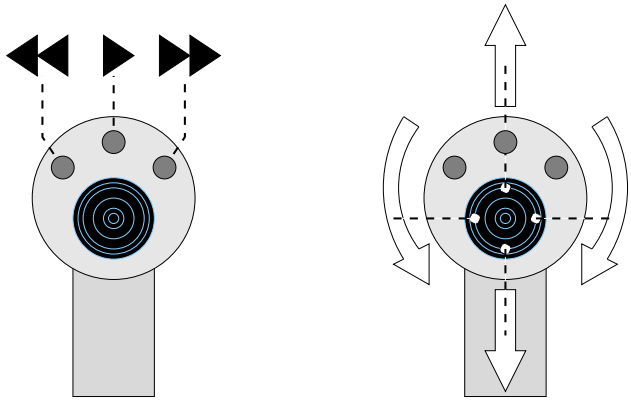


Figure 2.2: Interaction and navigation in the car crash application using the wand's buttons and joystick.

information that may have been calculated during the simulation, such as, for example, stress analysis results.
From a qualitative standpoint, the motion parallax effect provides a very intuitive way to inspect virtual objects. Where previously the user had to use an indirect interaction method (such as a mouse or keyboard) to orient the object on a desktop system, in the VE the user simply moves his viewpoint by walking around the object or moving the head. The navigation methods, controlled by the wand's joystick, are used to manoeuvre the CAVE through the virtual world to a location of interest. After that, the user simply walks around within the confinements of the CAVE for closer inspection. With some exercise, this interaction method is adequate for moving the viewer to any location of interest in little time.
The interaction requirements for this application were very simple. The biggest shortcoming we found with this application is its lack of expressiveness. For example; with the current implementation the "fast forward/rewind" type of interaction is not a practical interaction method to wind the visualization to *one particular* time step. The user needs to let go of the button precisely at the desired time step, which is easily missed if the animated playback is too fast. Of course, this could have been solved by having the left and right wand buttons advance only one time step at each button press, but that would then make it difficult to get a clearer idea on the behaviour of the simulation over longer periods of time. Again this problem can be solved, possibly by mapping short button presses to single time step advances and longer button presses to animated time step advances, or perhaps combinations of buttons pressed at the same time, etc., but imagine having to explain all this to the end-user. Moreover, the implementation of these interaction methods very quickly becomes far from trivial.
Clearly the user would benefit from interaction methods that allow more expressiveness than provided by the wand's buttons and joystick. The most obvious would be to extend the application with something that is similar to the graphical user interface (GUI) interaction methods we know well from desktop systems, i.e. menus, buttons, sliders, etc. Unfortunately, very few GUI toolkits exist for VR applications

and the ones that do exist are often very limited in their capabilities. Nevertheless, the availability of such a GUI toolkit would be a powerful way to obtain user-friendly and intuitive exploration environments. We will present a solution for this in section 4.3 (page 75), which describes a software architecture that allows existing 2D GUI toolkits to be used in VEs.

2.3 The Virtual Radiology Explorer

The use of VR technology in medical applications has already shown great potential in past studies [166, 197, 273, 274]. The applications developed in these studies have allowed a better understanding of complex anatomical structures. This has helped in areas such as clinical diagnosis, treatment planning, simulation and surgical intervention. However, a larger scale introduction into the medical society has met with resistance for various reasons. In this section we will look into the current methods that are used by radiology departments for the inspection of medical scans for diagnostic purposes and will investigate the use of VR technology in this respect.
The *Virtual Radiology Explorer* (VRE) project[†] was initiated to provide radiologists and physicians with a system that allows them to explore three-dimensional (3D) medical data sets, such as computed tomography (CT) and magnetic resonance imaging (MRI) scans, using VR techniques. One of the aims in this project is to provide intuitive, responsive and suitable interaction techniques through which the end-users will be able to efficiently perform diagnostic tasks.

2.3.1 The radiology department

To create an inventory of current methods that are used to interact with CT and MRI data sets, two radiology departments were involved: one at the University Hospital Utrecht (AZU, headed by prof. dr. P.F.G.M. van Waes), the other at Leiden University Medical Center (LUMC, headed by prof. dr. J.L. Bloem). Over a period of several days, both departments showed the current methods used to inspect CT and MRI scans for clinical diagnostic purposes, their use of 3D imaging techniques, and shared their ideas on future prospects in this regard. The radiology departments at AZU and LUMC use both CT and MRI scanners as their primary diagnostic instruments in cases where a correct diagnosis requires insight in 3D anatomy. The scans produced by most scanners consist of two-dimensional "slices" that each contain 512×512 gray scale pixels, each 16 bits in resolution. The number of slices made per scan depend on the spacing between slices, the thickness of each slice and the size of the structure of interest. Spatial resolution is measured by the maximum spatial frequency by which two lines close to each other in a line-bar pattern can be distinguished and is expressed as "line pairs per millimeter" (lp mm^{-1}). The spatial resolution for CT

[†]Funded by the *High Performance Computing in the Netherlands* (HPCN) Platform under ICES-KIS-2.

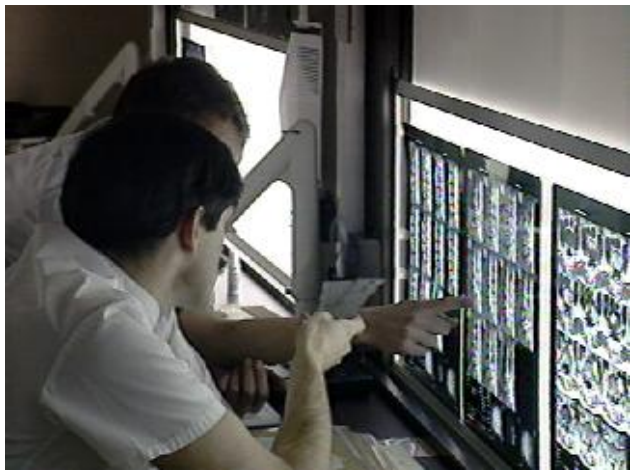


Figure 2.3: Inspection of hardcopy films on lightboxes in the radiology department.

scanners is 1 lp mm^{-1}, for MRI 0.5 lp mm^{-1} (1998). The spatial resolution of film used in X ray radiographs is 100 lp mm^{-1} [77].

Both departments transfer the scans both to a hardcopy machine, where high resolution transparency films are printed, and to a cluster of networked workstations (in both departments Philips *EasyVision* systems are used, running on Sun workstations). Along with other methods used for clinical diagnostics, a physician makes a diagnosis and proposes a treatment for a patient based on the images produced from these scans. Hardcopies of the scans on transparent film is the first and foremost method used for inspection. The hardcopy films are inspected using lightboxes in cases where one or two radiologists need to discuss a patient (see Figure 2.3). There is talk in both hospitals to switch over to "filmless" radiology departments, i.e. to a situation where only digital workstations are used for the storage and inspection of medical scans.

Beside the scanned image, the films contain information on the parameter settings of the scanner (including acquisition parameters), patient name, scan date, slice index and annotations that provide a reference for the radiologist on the orientation of the slices. AZU also uses overhead camera systems that enlarge specific regions of the hardcopy onto a television screen so that more people can see and discuss the images at the same time. Results of the inspection are voice recorded on quick-access mini audio tapes that are filed with the hardcopies into patient dossiers. These tapes are then sent to a department were they are converted into written text by typists or, in some cases, by speech recognition systems. For additional annotation, radiologists use red pencils to highlight regions directly on the film. Another method used for annotation is spoken-word-to-tape to refer to specific slices by index numbers and names of anatomical structures contained therein. The lightboxes often contain more than one set of films pertaining a certain patient. The material that is to be discussed is prepared by assistants on separate frames of the lightbox. These are brought into view by the radiologist by keying in a frame number on a keypad.

2.3.2 Desktop visualization

In cases where a side-by-side display of scanned slices is not adequate, Philips' *EasyVision* workstations provide a so-called "cine-loop" capability which allows radiologists and physicians to "flip" through a scan using a slider, as if it were a deck of cards. The slices can be displayed in one of three orthogonal orientations: sagittal (any vertical plane that divides the body into left and right parts), frontal (any vertical plane that divides the body into anterior [front] and posterior [back] parts) or transversal (any horizontal plane that divides the body into superior [towards the head] and inferior [towards the foot] parts). A user-defined oblique orientation is obtained by drawing lines on the slices to denote the position and orientation of intersection planes. Beside this cine-loop functionality, the scans can be displayed side-by-side as on the hardcopy films. In this case the number of slides in a row is set interactively, allowing the slices to be viewed at different magnification, as well as with user settable contrast and brightness of the displayed images. The *EasyVision* workstations are not used for annotation or reporting purposes. The cine-loop feature of the *EasyVision* workstation is the most often used function. Although the software provides 3D rendering (see Figure 2.4), both volume and surface rendering, these capabilities are used sparsely because radiologists find them too complicated to use and, as a consequence, feel it takes too much time to produce useful results. At AZU, 3D rendering is often delegated to the local imaging research group (which has much experience with 3D visualization), while at LUMC *EasyVision* is used more for this purpose.

The *EasyVision* system provides several tools to process a scan before it is visualized in 3D. One such tool allows irregularly shaped areas in a stack of slices to be discarded

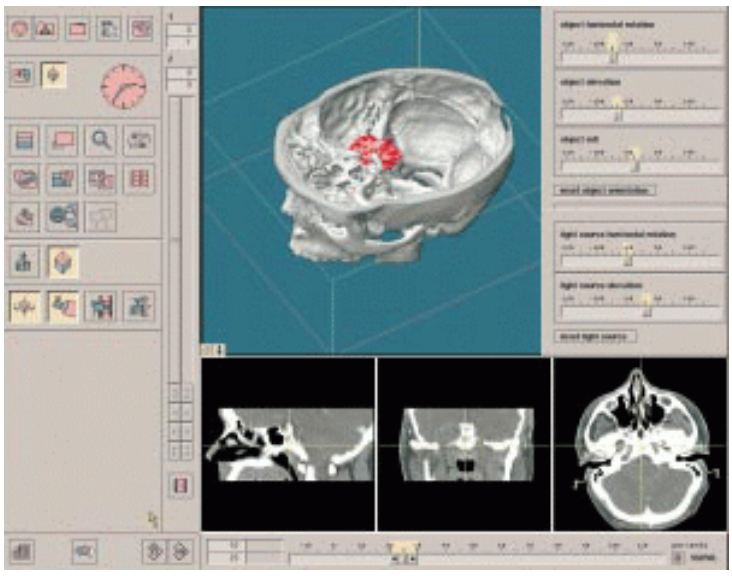


Figure 2.4: Philips Medical Systems' EasyVision *showing an example of 2D and 3D visualization.*

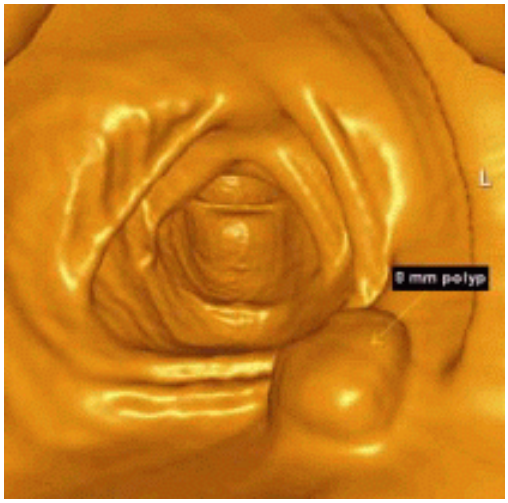


Figure 2.5: Philips Medical Systems' 3D EndoView. *This example shows an 8 mm polyp located in the ascending colon [199].*

from the 3D rendering so that specific anatomical structures can be highlighted. In addition, Philips provides several extensions for the *EasyVision* system that provide virtual "walkthrough" capabilities (for example, see Figure 2.5).

2.3.3 VRE objectives

Clearly, three-dimensional representations of medical scans are not common practice in present day radiology departments, despite the fact that CT and MRI scans are inherently volumetric. There is a number of reasons for this. First; the technology to reconstruct these data sets to 3D representations in a timely fashion, of sufficient quality that can *also* be manipulated interactively has just recently become available at an affordable price. Indeed, both departments *have* desktop workstations that allow them to create 3D representations of anatomical structures. Little use is made of this, however, mostly because the radiologists do not know how to use the workstations and because it takes too much time to obtain acceptable results. However, there is a strong interest to use the 3D information that is available, as (for example) illustrated by liver surgeon Rory McCloy in Nature, March 2002: "I spend my life looking at 60 slices of salami. [...] I'm trying to do a 3D operation with 2D images" [180]. More important, however, is that the two-dimensional diagnostic methods that are currently used *work* and therefore there is little desire with the radiologists to use new methods that have not yet proven their value. Therefore, if a paradigm shift such as proposed by VRE is to succeed, the transition for its users should be made as easy as possible. Interaction should therefore be a key concept in its design.

2.3.4 Visualization and interaction methods in VRE

The visualization and interaction methods for VRE should make the transition for physicians and radiologists from their conventional diagnostic methods to a virtual environment as comfortable as possible. The interaction mechanisms described here are primarily intended for use in immersive projection based VR systems such as the CAVE and ImmersaDesk. However, most of the methods described here should still be applicable to other types of VR systems. Additional comments on differences and potential problems are explicitly noted where appropriate. To make the transition for users of VRE as simple as possible, different levels of functionality should be offered, ranging from the "conventional" and "well known" methods that are currently in use, to new methods as proposed by the VRE project. As a basic functionality, VRE should therefore offer radiologists functions that mimic the lightbox and cine-loop as are used in daily clinical diagnostic tasks. Beside that, the new functions that VRE can offer are 3D techniques with new capabilities such as stereoscopic rendering, virtual endoscopy, virtual colonoscopy and others.

Volume visualization

Visualization methods for the representation of 3D scalar lattice volumes are usually separated into two groups: (direct) volume rendering and isosurface modeling [69,72]. Volume rendering is a technique that is based on ray casting [79]. All scalar values in the 3D lattices are traversed and treated as volumetric elements that contribute both colour and opacity to a virtual ray of light that travels through the data set towards the viewer's eyes [146,240]. The contribution of colour and opacity is defined through a transfer table that maps a scalar value onto a colour and opacity value. Using this table, structures that are not of interest to the user can be made transparent by mapping its associated scalar values to translucent opacity values. Likewise, interesting structures can be emphasized by mapping the associated scalar values to brighter colours. The definition of a transfer table that yields good visual results can be time consuming, or worse; close to impossible. In most cases, the transfer table is used to define a colour and opacity gradient over a range of scalar values. The assumption is that structures of interest consist of scalar elements that have neighbouring values. In the case of medical data sets, however, this assumption depends on the data acquisition method that was used in obtaining the data set. For example; in CT scans the scalar values (or "Hounsfield units" as they are called in the case of CT) are a representation of the attenuation of an X ray beam through the human body [77]. As this value varies with the density of the tissue (the denser the tissue, the higher the value), similar tissue structures will yield similar values. Defining a transfer table that accentuates the different types of tissue is, in this case, relatively trivial. In MRI scans, however, the scalar values represent the radiofrequency (RF) energy emitted by the nuclei of hydrogen atoms (free or attached to other molecules) in a strong magnetic field after excitation by a microwave radio signal [182]. As most parts of the human body consist of fat and varying concentrations of water (and because these molecules contain hydrogen atoms), the scalar values in the data set are mainly a representation of concentrations of hydrogen. Similar concentrations of hydrogen are not always part of the same physiological structure. This makes the definition of a transfer table far more difficult.

In isosurface modeling, an intermediate representation is first computed that consists of geometric primitives; usually triangles [151]. These triangles represent surface patches through lattice elements of the same value, resulting in a constant value contour surface. As with volume rendering, the definition of this constant, so-called "threshold" value relies on the property that scalar elements that are part of the same structure are of approximately the same value. Because isosurfaces only represent structures at the same scalar value, it can in some cases be difficult to relate the isosurfaces to the structure as a whole. To compensate, multiple isosurface models are often computed at different threshold levels that are then rendered together in the same scene, with different colour characteristics to be able to tell them apart and different opacity settings to reveal otherwise occluded structures.

Interactive volume visualization

Additional challenges in successfully applying these visualization methods in an interactive virtual environment are that (1) the time required to *compute* the visualization is small enough that the user can change parameters of the visualization method and see the results quickly, and (2) the time required to *render* the resulting visualization is fast enough to allow interactive exploration in a VE. The first challenge is most apparent with isosurface modeling where an intermediate triangle representation needs to be calculated before this representation can be rendered. The second challenge is determined by the performance characteristics of the graphics hardware.

In the case of volume rendering, the 3D lattice structure has to be completely traversed each time the position or orientation of the user changes and/or when the user changes the transfer table. Various implementations of specialized software and hardware have been developed that allow volume rendering to be used in virtual environments at interactive speeds [37, 102, 133, 191]. Most of these methods use 2D or 3D texture mapping and transfer table lookup techniques that are often accelerated in hardware. Using these techniques, the resulting images can be rendered directly based on the current position and orientation of the user and the defined transfer table. A limiting factor on rendering performance is the "fill rate" of the graphics hardware; the rate at which pixels can be drawn into screen memory. Fill rate is usually measured in millions of pixels per second (Mpixels/s) and is directly dependent on the hardware architecture (i.e. the bandwidth of the memory bus and the ability of the graphics hardware to saturate this bandwidth). In particular in the case of volume rendering, fill rate limits the frame rate when the area that is covered by the resulting image on screen increases.

The intermediate triangle representation used in isosurface rendering is independent of the position of the user and therefore only needs to be calculated once when the threshold value has been set or changed. The triangle representation must then be rerendered for the current position and orientation of the user. However, the time required to render a new frame is directly related to the number of triangles in the isosurface. The number of triangles that a hardware graphics interface is capable of rendering in one second is a popular (but inconclusive) measure to characterize its performance. For example; the InfiniteReality2 graphics pipeline used in the SARA CAVE is capable of rendering 11 million triangles per second [217]. Suppose the virtual environment needs to maintain a frame rate of at least 20 *stereoscopic* frames per second for a particular application. This means there is only $\frac{1}{40}$th of a second to render each frame which in turn implies there should be no more than 275,000 triangles in the complete scene. Although this may seem like much, consider that it may often be necessary to visualize *multiple* isosurfaces, at different threshold levels, so that anatomical structures can be viewed in relation to others. In these cases it may be necessary to reduce the number of triangles in an isosurface but only if the original geometry can be maintained. Decimation is one technique to reduce the number of triangles in an isosurface triangle mesh while preserving the original topology and forming a good approximation to the original geometry [210].

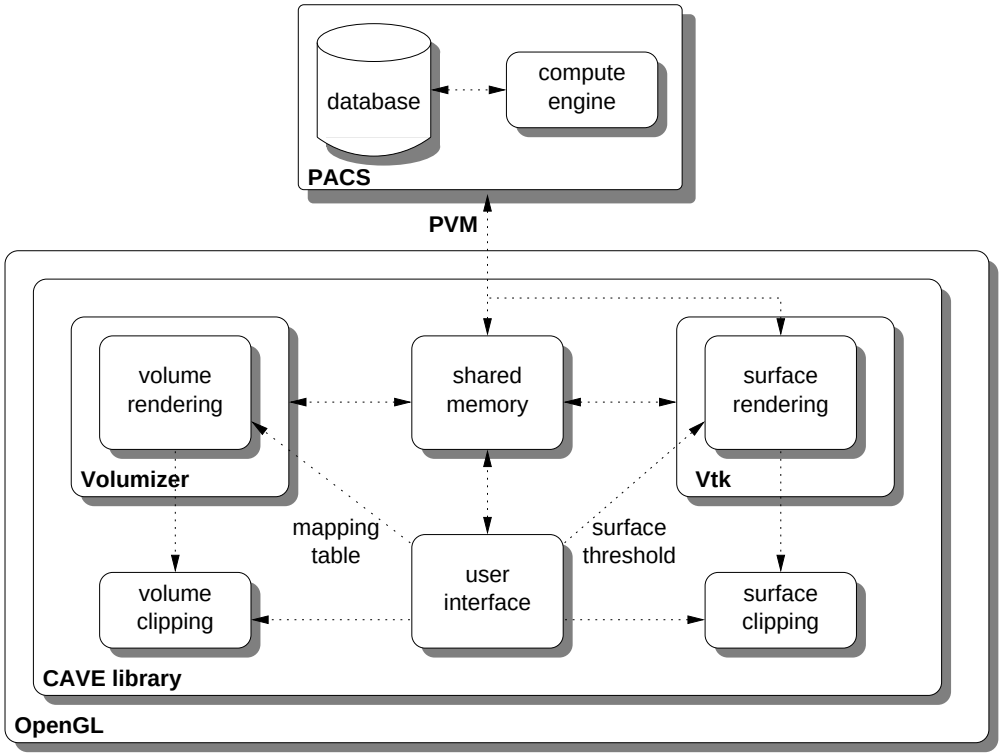


Figure 2.6: Architecture of the VRE application.

Although decimation helps in obtaining an isosurface contour that can be rendered at interactive speeds, the additional time required to compute the isosurface can be substantial. Considering that the calculation of an intermediate triangle representation for isosurface rendering operates on neighbouring lattice sites, the total execution time can be decreased by decomposing the 3D lattice into subdomains and perform the isosurface extraction in parallel, on multiple processors [202].

2.3.5 Implementation

The architecture of the VRE application is shown in Figure 2.6. VRE is implemented in C++ [227] and runs on all Unix operating systems that support CAVELib and OpenGL. CAVELib is used to hide the intrinsic details of the display technology (such as the placement of the projection displays) and interaction devices (wand, joystick and buttons) [249]. OpenGL is used as the graphical rendering library [261]. For volume rendering we use Silicon Graphics' OpenGL|Volumizer, a specialized library that supports hardware accelerated volume rendering on Silicon Graphics hardware[‡] [191]. Surface modeling is performed by the Visualization Toolkit (Vtk) [106, 209]. The volume and surface representations can be individually clipped to hide parts of

[‡]This limits the use of volume rendering to Silicon Graphics systems only. The remaining functions can still be used on other systems. "CAVORE", a CAVE volume rendering package developed by Anton Koning (SARA) for the VRE project, can be used on all systems that support 3D texture mapping [133].

the renderings. This allows hybrid representations that consist partly of a volume rendering and for the other part of a surface rendering.
A simple networked database interface has been implemented that emulates a Picture Archiving and Communication System (PACS) as is used in most radiology departments to interface medical scanners to visualization front-ends. The database contains patient data, stored on a remote IBM SP2 system located at SARA, which is accessed by a user interface from within VRE. VRE and the PACS server communicate using PVM [229]. Beside a database function, the PACS component is also equipped with a computing function that is capable of performing parallel isosurface modeling, which will be described below. If the user wishes to visualize an isosurface representation of a specific dataset, the data set identifier and the desired threshold level are sent to the computing engine which then extracts the isosurface contour and sends the results back to the VRE application in the form of a geometrical representation.

Parallel isosurface modeling

The parallel isosurface modeling has been implemented using Vtk. The visualization pipeline is shown in Figure 2.7 and works as follows. The input data set is decomposed over the available processors using "standard" domain decomposition; each processor $p_{0\leq i<P}$ reads $\lfloor N/P \rfloor$ slices (where N is the number of slices in the input data, P the number of processors and $P \leq N$), unless $N \bmod P \neq 0$ in which case processors $p_{i<N \bmod P}$ read $\lfloor N/P \rfloor + 1$ slices. Note that we assume with this decomposition method that the workload for each processor will be the same when they are given equal shares of slices. As this assumption depends largely on the data contents and the threshold value selected for the isosurface, the decomposition method used here will not always result in equal workloads.

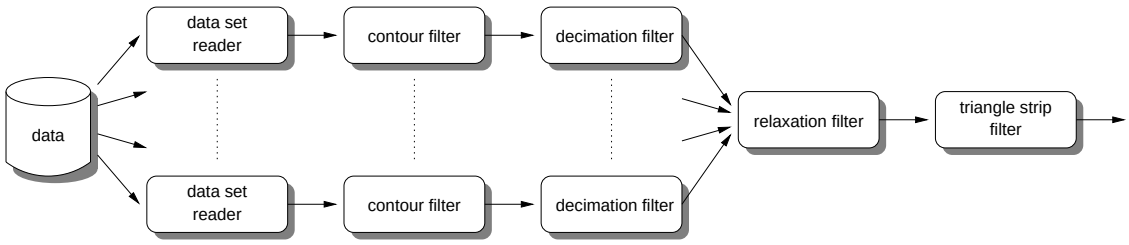


Figure 2.7: Visualization pipeline used for parallel isosurface modeling.

Each processor proceeds by modeling an isosurface for its local domain using a marching cubes contour filter [151]. The number of primitives in the resulting triangle representation is then reduced using a decimation filter and concatenated to form a complete isosurface [210]. If the concatenated isosurface produced at this point would be used for rendering by OpenGL, an unfortunate side effect of the distribution over multiple processes can result, as illustrated in Figure 2.8. If the local isosurfaces at the boundaries of neighbouring local domains are curved differently, the isosurface patches will be shaded incorrectly. The shading method used in OpenGL is Gouraud

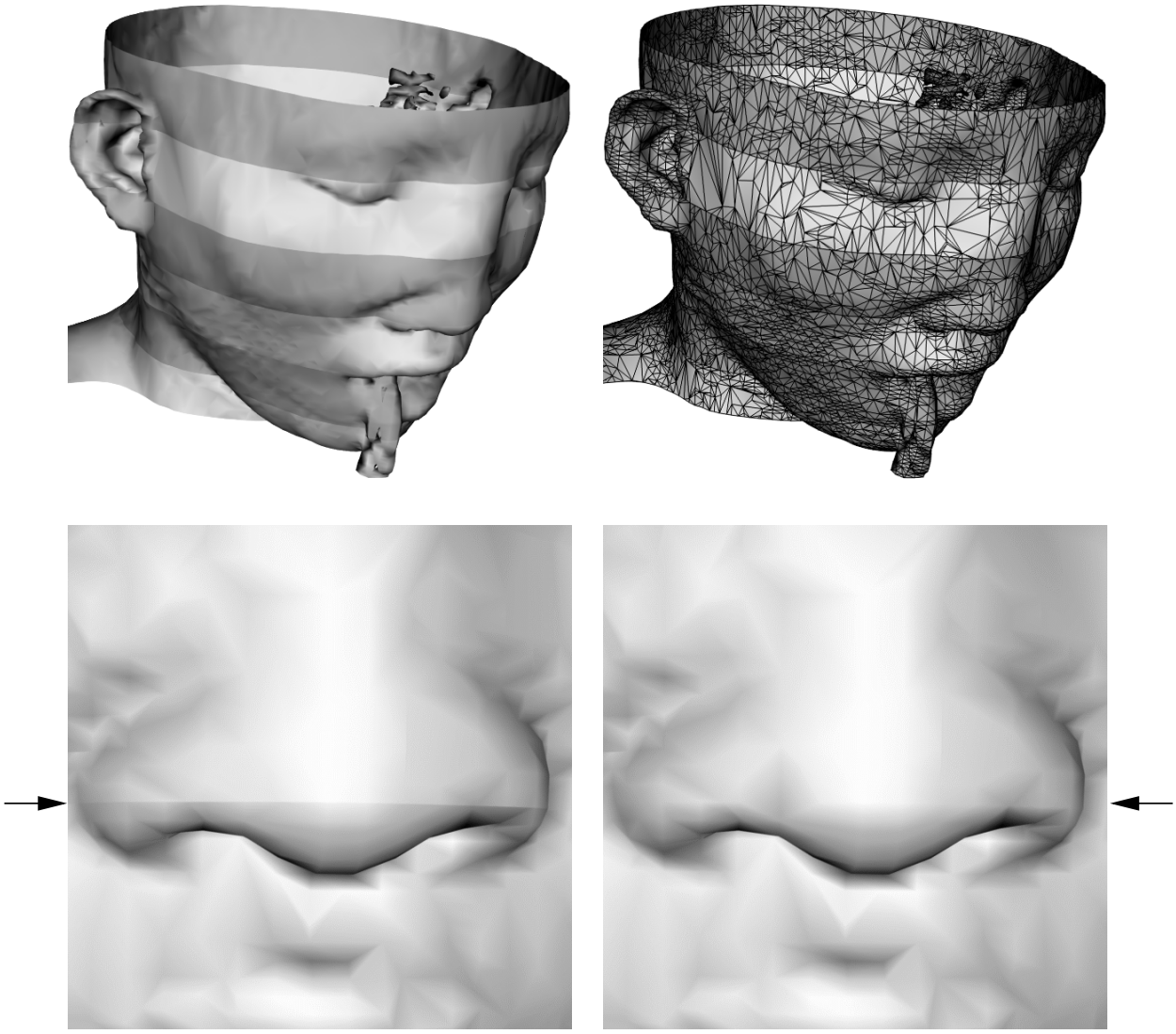


Figure 2.8: Parallel isosurface extraction; domain decomposition (top left); for each subdomain, an isosurface is extracted and the resulting triangle mesh is decimated (top right); lighting artefacts appear on curved domain boundaries caused by ill-defined normal vectors (bottom left); relaxation of the triangle mesh corrects these artefacts (bottom right). The two bottom images have been colour-enhanced to show detail.

shading which requires that a normal vector is defined for each vertex in a surface patch [89]. The normal vector in each vertex is determined by first determining all faces that share the vertex. The normals of all adjacent faces are then averaged to get the vertex normal. At local domains, the normals of adjacent faces on neighbouring domains are not available and the vertex normal is averaged over locally adjacent faces only. The result is an ill-defined normal vector which is most conspicuous on

strong curved boundaries. In our implementation, this problem is solved through a relaxation filter on the concatenated isosurface, implemented as an additional decimation step that is set to only merge co-planar surface patches and recalculate vertex normals. The resulting isosurface will have correctly defined vertex normals and, at the same time, the number of triangles is reduced even further.
Finally, a triangle strip filter is used to convert the isosurface into a representation that requires less space to store and that can be rendered at higher efficiency by OpenGL [261]. The end result is transfered to VRE for rendering.

Interaction methods

Interaction in VRE is done primarily with the wand, including its buttons and joystick. In addition, a simple but effective menu system has been built that provides access to all of VRE's functionality. Options on the menu are selected by pointing at one of the items and clicking a wand button. A virtual pointer is rendered from the front of the wand to provide visual feedback; seeing the pointer intersect with the menu as well as the highlighting of the selected menu items aids in the menu selection process. Again; the virtual pointer also helps in unambiguously identifying interesting structures to other users as described in section 1.5.2 (page 19). The menu provides access to the patient data sets stored in the PACS database, enables/disables visualization options and allows various visualization options to be set interactively, including object scaling, rotation, translation, the desired isosurface modeling threshold, the sampling rate used for volume rendering and storing/retrieving transfer tables.
The joystick on the wand is used in two modes, as selected from the menu. In the first mode, the wand is used as in most CAVE applications; it moves the position of the CAVE towards the direction where the front of the wand is pointing, at a velocity proportional to the amount the joystick is pressed forward or backward. Sideways pressure on the joystick is used to rotate around the center of the CAVE, around the y-axis (which in the CAVE points upwards) (see also Figure 2.2, page 28). Although this so-called "navigation" takes some getting used to at first, this method of navigation quickly becomes "natural". In the second mode, the wand is used to transform the visualized objects: it can be used to scale, translate or rotate the object as specified by a selected transformation in a menu option. For purposes such as virtual endoscopy, the user can use the scaling transformation to "blow up" isosurfaces of structures until they are large enough to inspect the insides of the structure. This scaling is mandatory for this feature: although it is possible to view the insides of rendered object merely by "sticking ones head in", this can be a most unpleasant strain on the eyes as they have to accommodate on structures that are too close [99]. Note that volume rendering is not suitable for this type of interaction due to the nature of the direct rendering algorithm; the resulting images show up as big blobs of pixels from which no structure can be discerned. Also, on most of today's hardware, scaling up the volume would quickly reach the fill rate limitations of the graphics hardware resulting in low frame rates and sluggish responsiveness.

Clipping is the VRE equivalent of the cine-loop capability used on medical workstations. It allows the user to cut away sections of a visual object (isosurface, volume or both individually) using a plane that reveals the inner parts of a dataset that would otherwise be obscured. Once activated via the menu, the clipping plane is attached orthogonally to the forward-pointing wand, at a constant distance, and follows the position and orientation of the wand. Interaction is very intuitive: the user only needs to move the wand into the visual representation of the scan to inspect its interior. Clipping is implemented using OpenGL's standard clipping mechanism and is fully hardware accelerated.

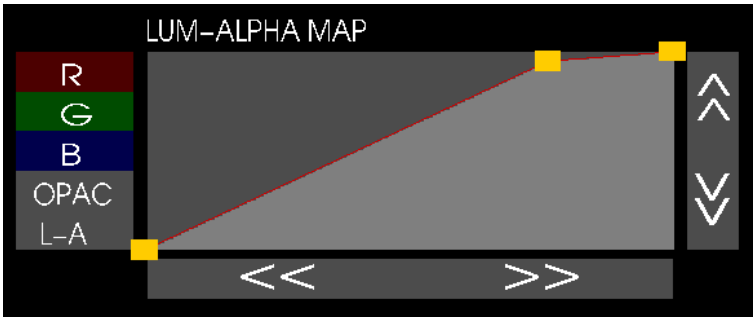


Figure 2.9: Transfer table editor used in VRE. Scalar values on the horizontal axes are mapped to colour properties on the vertical axes. The editor supports interactive definition of separate mappings for red (R), green (G), blue (B), opacity (OPAC) and luminance-alpha (L-A). Shown here is a luminance-alpha mapping with one additional control point.

The interactive definition of the transfer tables for volume rendering is done using the editor shown in Figure 2.9. The rectangular buttons on the left allow separate transfer tables to be defined for red, green, blue, opacity (or alpha) and luminance-alpha. Luminance-alpha can be regarded as a mapping where red, green, blue and opacity values are identical, resulting in dark/transparent colours for low scalar values to luminous/opaque colours for high scalar values. The buttons marked with arrows on the right and at the bottom allow the transfer tables to be shift and scaled. By default, a linear "luminous-alpha" mapping is defined from dark/transparent for low values (shown by a square in the lower left corner) to luminous/opaque for high values (shown by a square in the upper right corner). The linear mapping between these values is shown by a diagonal line between the two. This mapping can be altered by introducing new control points anywhere on the diagonal line and moving the control points over the window.

2.3.6 Experiences

We organized two proof-of-concept demonstrations for a number of radiologists and physicians from the hospital; one took place in the CAVE at SARA, Amsterdam, the other in the radiology department of the Leiden University Medical Center (LUMC,

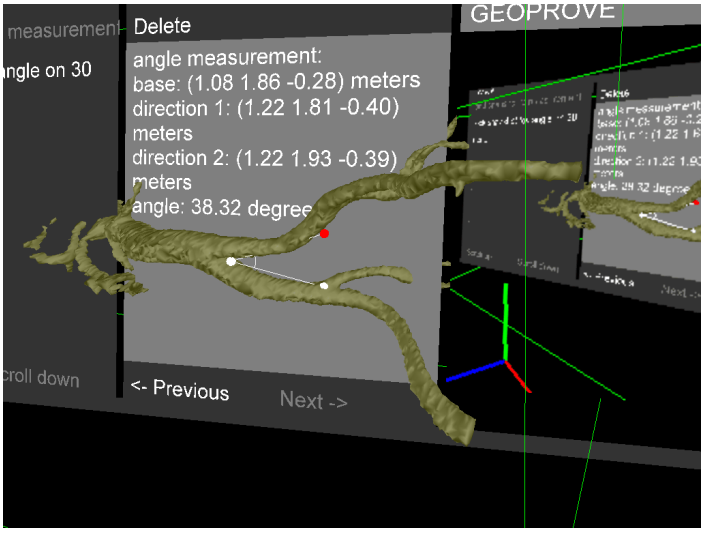


Figure 2.10: Surface rendering of the abdominal aorta. GEOPROVE *(see section 4.5) is used to measure the angle of the bifurcation (see also colour reproduction on the back cover).*

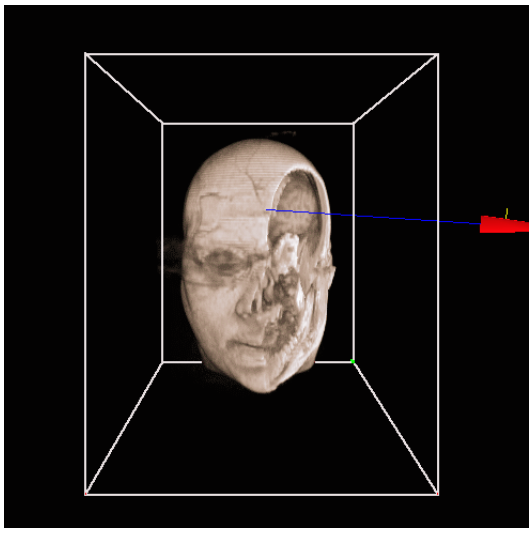


Figure 2.11: Interactive oblique clipping of a volume rendering of a human head MRI scan (image created by Anton Koning, SARA).

Leiden). For the latter we installed an IDesk projection system in the department with a Silicon Graphics Octane as the computing and graphics hardware. The data sets used in the demonstrations included patient scans provided by the radiologists as well as scans obtained from other sources, such as the Visible Human data sets [2, 174, 223]. The VRE application showed how patient data stored in a database on the IBM SP2 at SARA could be loaded via a network connection and visualized using volume and surface rendering techniques (see Figures 2.10 and 2.11). In addition, the system demonstrated how isosurface modeling could be executed on the IBM SP2, the results of which would then be transfered back to the visualization front-end for rendering.

During an evaluation meeting it became clear that the radiologists and physicians regarded the VRE environment as a useful instrument for educational, demonstration and communication purposes. However, they agreed that the quality of the visual representations are insufficient for diagnostic purposes unless applied to *very* specific situations in which the visual analysis of three-dimensional structures would be required. The foremost problem was the lack of texture and detail in the visual constructs generated by VRE. The most important reason for this is a hardware limitation of the graphics pipelines that were used, both in the Onyx2 of the CAVE and in the Octane; most of the data sets used for the demonstrations had to be downsampled and reduced in scalar resolution to fit in the available texture memory. Furthermore, some of the interaction mechanism provided were regarded as difficult to use, most specifically the definition of colour transfer tables in volume rendering (which in some cases could be solved by providing presets), determining proper isocontour levels for surface rendering (which in some cases could be solved by providing image

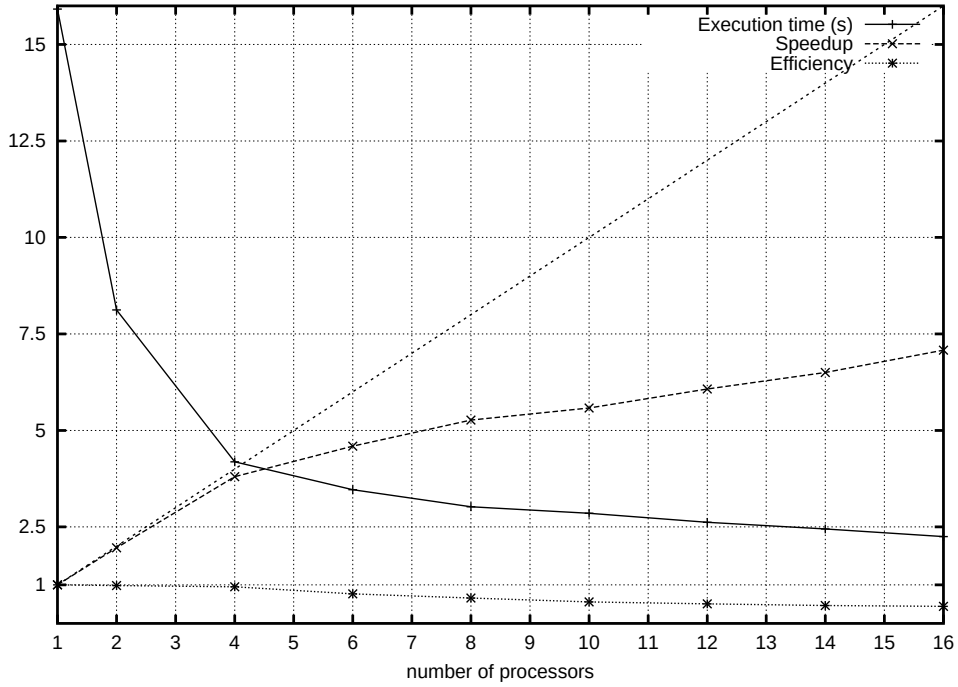


Figure 2.12: Execution time (in seconds), speedup and efficiency of the parallel isosurface engine on a high resolution version of the data set shown in Figure 2.8.

histograms) and the techniques for the manipulation of the presented constructs (i.e. rotation, scaling, translation versus navigation in the VE).
In addition to this, there are also practical reasons why radiologists will not adopt this kind of technology right away. Radiology departments do not have the funds, manpower or space to house and maintain a CAVE installation. Nor will they be willing to spend time to travel to a CAVE installation elsewhere. Smaller systems like the IDesk come a little closer to a solution as these can be used in the radiology department. Still, these systems are too expensive. In Chapter 3 (page 51) we describe the construction of low-cost VR systems based on off-the-shelf computing hardware.
An addition to the VRE system that would make the system more useful was found to be a capability that allows measurements to be taken from the visual presentations. Physicians often need information on the size of certain anatomy and/or pathology in order to prepare a surgical procedure. Obtaining measurements from visual representations in a VE is an area in which very little research has been done. We took up that challenge and present an architecture to do measurements in VEs, called GEOPROVE, in section 4.5 (page 86).
Figure 2.12 shows the performance characteristics of the parallel isosurface engine while modeling the isosurface of the skin from a CT scan data set of 256×256 pixels per slice, 94 slices, 16 bits per pixel (this is a high resolution version of the data set shown in Figure 2.8 and with the same threshold value). This figure shows that the extraction of isosurfaces on multiple processors does result in a reduced execution time but that the efficiency of the parallelization is far from optimal. As already noted

earlier, this is caused by the domain decomposition method that was used. Indeed, performance measurements of the same algorithm on uniform data, resulting in an equal workload on all subdomains, show an almost linear speedup (data not shown). To increase efficiency of the program, methods should be introduced to balance the workload over the available processors. Because the performance of the current implementation was considered acceptable for our purposes we have not pursued these methods. Various methods could be used to improve the workload balance to obtain better efficiency such as alternative data decomposition methods or through farming of the data in small portions over the available processors.

2.4 Diffusion and flow limited biological growth

In our research group there is a strong interest in the study of biological systems [111–121]. Computational simulation models play a fundamental role in the study into the behaviour of these systems. What is frequently missing are methods to analyse the results of these simulation models that help in their validation. Frequently, automated analysis of the simulation results is difficult due to the unavailability of suitable algorithms or, in cases where algorithms do exist, the computational requirements are too high to perform an effective analysis. In the following case study, we address a computational model for the study of marine sessile organisms, such as sponges and stony corals. We describe an immersive visualization environment that is used for the interactive visual analysis of simulation results.

2.4.1 Background

In the development of many biological systems, the distribution of chemical agents and nutrients plays a fundamental role. For filter-feeding marine sessile organisms, such as stony-corals, the growth process is affected by the distribution of suspended material in the external environment. From the biological literature it is well-known that water movement may have a strong impact on the shape of stony-corals. It is often possible to correlate growth forms of stony-corals with the amount of water movement. Compact growth forms are generally found under conditions with a large exposure to water movement, while the growth form changes gradually into a branching shape when the amount of water movement decreases. Figures 2.13 and 2.14 show two growth forms of the stony-coral species *Pocillopora damicornis*. The compact form in Figure 2.13 originates from an exposed site and the thin-branching form in Figure 2.14 was collected from a sheltered site.
Our research group has studied the effect of hydrodynamics on a very simple type of growth process, viz. growth by aggregation. In this model, aggregation proceeds by the accumulation of a “nutrient”. The nutrient distribution is modeled using a Lattice Boltzmann model of transport. The aggregate absorbs the nutrient and the amount absorbed determines the local growth probability. We have carried out simulations of growth processes (aggregation processes) in which an aggregate consumes nutrients

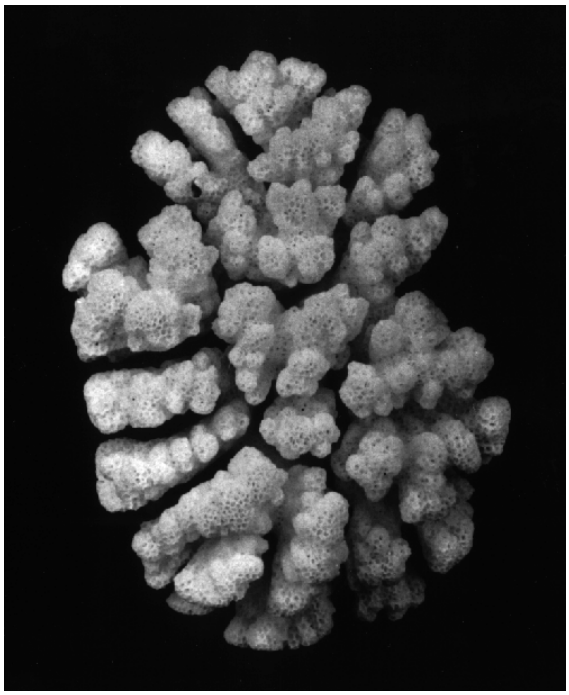


Figure 2.13: Growth form of the stony-coral Pocillopora damicornis *originating from an exposed (to water movement) site.*

Figure 2.14: Growth form of the stony-coral Pocillopora damicornis *originating from a sheltered (to water movement) site.*

from its environment and where nutrients are dispersed by a combined process of flow and diffusion [121]. The effect on the aggregate caused by different rates of fluid flow and nutrient dispersion is investigated.

2.4.2 Analysis of simulation results

The data resulting from these simulations includes the growth of the aggregate over time, the dispersion of nutrients around the aggregate, the absorption of nutrients on the surface of the aggregate and the velocity of flow around the aggregate. Growth in the data is encoded as a three dimensional volume of grid nodes $V(x,y,z)$ where $V(x,y,z) = t$ denotes that the grid node at (x,y,z) was aggregated at time step t (and $V(x,y,z) = 0$ in grid nodes where no aggregation took place). Originally, the only method to obtain insight in the results of the simulation was through visualization of the generated data sets and visually comparing these with existing coral structures. A special purpose software package was implemented to obtain surface models of the generated structures at each growth step T by extracting an interpolated isosurface from the volume through all grid nodes where $V(x,y,z) = T$, which were then visualized on a graphical workstation. This resulted in a 3D surface representing the shape of the aggregate. However, the complexity of the aggregates was such that the generated surface models were too big to allow interactive exploration. In such cases it

was often necessary to generate animations on video which took well over a week to produce. In addition, the end results were inherently non-interactive which impedes exhaustive exploration.

2.4.3 Interactive exploration in Virtual Reality

To allow the simulation data to be explored interactively, we have built an environment that allows the simulation data sets to be explored inside a CAVE. This environment supports interactive visualization of surface models of the aggregate at any time step, animated playback of the development of the aggregate from start to finish, surface models of the nutrient distribution around the aggregate and colouring of the aggregates based on absorption.
For our initial experiments towards the development of an exploration environment we have taken a simulation model for the investigation of diffusion and flow limited biological growth. We have used data sets resulting from simulations of growth processes (aggregation processes) in which an aggregate consumes nutrients from its environment and where nutrients are dispersed by a combined process of flow and diffusion. Details about the simulation model are given elsewhere [121]. As an example, the effect on the aggregate caused by different rates of fluid flow and nutrient dispersion is investigated. The data resulting from these simulations includes the growth of the aggregate over time and the dispersion of nutrients around the aggregate. This model has been used as a simple model for coral growth [121].
Our interest is to compare the simulated structures with structures found in nature, the investigation of the complex geometry generated by the simulation and the flow fields around it, the behaviour of tracer particles released in the flow field, the location of nutrient absorption points, and the location of pressure fields causing hydrodynamical forces on obstacles. A crucial issue in the development of morphological simulation of growth processes is the ability to compare simulated growth forms with the actual objects. For this reason we have compared data sets of actual objects to the simulated growth. The data on the actual objects were obtained from CT scans made of some samples of the stony-coral *Pocillopora damicornis*. These CT scans consist of slices that each contain 512 by 512 16-bit gray scale values, where the number of slices depends on the length of the object and the number of rotations made by the scanner. The model shown in Figure 2.15 was reconstructed from 30 CT slices from which a surface contour was generated using an isosurface extraction algorithm in the Visualization Toolkit [209]. The CT scanner used for this scan was a Philips Tomoscan SR7000. The relatively low number of slices results in a decreased resolution in one of the principal axes which poses some problems in the reconstruction of a model that should be accurate enough for quantitative comparison.
Although our exploration environment provides methods by which the data sets can be explored visually, an important aspect in the development of any simulation model is its verification against the system that is modeled. A major problem in the quantitative comparison of the simulation results with actual phenomena is that in many cases there is no single discriminative feature by which they can be differentiated.

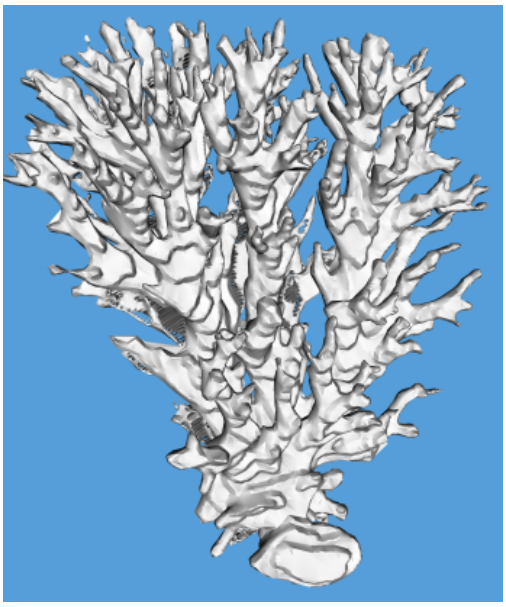


Figure 2.15: Surface reconstruction of a CT scan of Pocillopora damicornis.

In our test-case for example, a property such as the fractal dimension [155] gives some insight in the global resemblance of different structures but is inadequate in describing the quality of the simulation model as only a limited aspect of the overall morphology of an object is captured. Therefore, it is often more suitable to obtain measurements on multiple properties in local areas of the data sets that together form a discriminative measure.

In case of the growth model we wish to compare the shape of the resulting structures to those found in nature. In previous work it has been demonstrated that morphological properties such as for example the thickness of branches and the shortest distance between neighbouring branch points ("branch spacing") provides relevant biological information and can be used to compare simulated with actual growth forms [118, 119].

In addition, when comparing simulated coral objects with actual corals, the comparison procedure should be non-destructive to the real coral as most of these are valuable and irreplaceable specimens. However, this makes many measurements difficult if not impossible since the complex shape of these structures prohibits the use of instruments that may damage the coral. One possible solution is to acquire a sufficiently accurate three-dimensional scan of the coral. Since conventional photographic or laser scanning techniques are only suitable for obtaining surface models of objects which have no obstructing components, these devices are unsuitable for scanning complex and irregularly structured objects such as corals. Fortunately, we have obtained digital 3D data sets of a number of corals which we can use for our purposes through the assistance of the Radiology department of Leiden Academic Hospital, who have graciously offered to scan the corals with a computed tomography (CT) scanner.

Although the properties we want to measure could be obtained automatically using

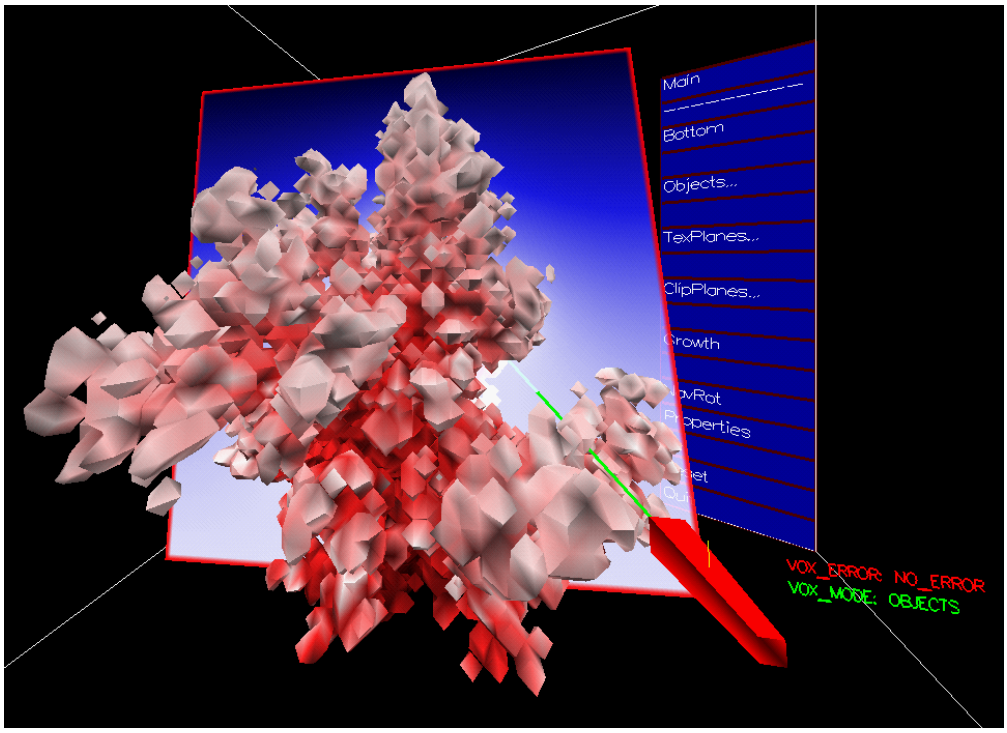


Figure 2.16: CAVE application for the exploration of aggregation processes. (see also colour reproduction on the back cover).

data analysis techniques, this often requires designing and implementing specialized algorithms that are dedicated to the specific task. Quite often these techniques rely on heuristic algorithms that are difficult to design, implement and control. An interactive environment equipped with a system that allows measurements to be taken from the visualizations that are rendered in the virtual environment can provide the techniques needed to acquire quantitative properties from data sets which would have been difficult to obtain otherwise. We have designed and implemented a system to support this, called GEOPROVE which will be described in detail in section 4.5 (page 86).

2.4.4 Exploration in the CAVE

We have built an interactive exploration system that allows the data sets, generated by the simulation, to be explored inside the CAVE located at SARA. Within this interactive exploration system, surface models were used for visualizing the growth of the objects, a tracer distribution model for studying tracer distributions about complex objects and for measuring the degree of absorption of tracers at the objects, methods for visualizing the flow field around the objects, and methods for sectioning the objects which enables us to study the addition of material during the growth process.
Figure 2.16 shows the result of a simulated aggregation process. The aggregate emerges in an environment where nutrients are mainly dispersed by diffusion. In

this case an irregular branching aggregate is formed. The colour of the object represents age; from dark red for "old" parts, to white for "young parts". The nutrient distribution around the aggregate is visualized using a blue-white gradient, where blue indicates a high and white a low nutrient concentration. The exploration environment facilitates the interactive inspection of these gradient planes by allowing the plane to be moved through the simulated structure over all three principal axes.

2.4.5 Discussion

Using the developed exploration environment we have investigated simulated results of various experiments in which the influence of hydrodynamics on the growth process was varied. In addition, we have been able to compare these results with CT scans of actual stony-corals. Using these CT scans, a more flexible comparison of the simulated structures to those found in nature is now possible. The qualitative comparison showed that both the simulated growth forms and the actual stony-corals show a similar tendency: when the influence of hydrodynamics increases, both simulated and actual forms exhibit an increase in compactness. This observation corresponds to the observations reported in [121]. The exploration system has also shown differences which were not detected before: when comparing the CT scans to the simulated results it was found that there is a difference in the branching patterns. Especially in the compact aggregates branches tend to fuse (anastomosis) easily, while this phenomenon was not detected in the CT scans. This observation indicates a difference between the actual growth and the simulation model.
Our main finding is that the use of this exploration environments in the CAVE allows us to study the effect of flow on the nutrient distribution far more easily than was previously possible. The environment allows us to study the morphology of the aggregates simply by walking around the presented object. Using the CAVE's "wand" we are able to explore the growth of the aggregate over time, and the dispersion of nutrients around the aggregate.

2.5 Summary and conclusions

In this chapter we described three interactive exploration environments: one for the exploration of car crash simulation results, one for the exploration of 3D medical data sets and one for the exploration of simulation results of a diffusion and flow limited biological growth modeling simulation. Our aim was to obtain a better understanding of the different issues involved in the design, implementation and use of these interactive exploration environments.
To obtain usable environments in terms of the requirements described at the start of this chapter (i.e. quality of presentation, rapid frame rate, intuitive interaction and real-time feedback), we had to make compromises. Some of these have resulted in minor nuisances. For example; in the car crash environment, the use of display lists increases application startup time but results in higher frame rates and increased

responsiveness. Also, the parallel isosurface modelling method used in the VRE environment results in models that can be displayed at high frame rates but at the cost of increased response time. Other compromises are far more serious; the reduced image quality as a result of both the down-sampling and the reduction of scalar resolution of medical data sets (to compensate for hardware limitations), results in visual representations that medical experts find insufficient for diagnostic purposes. Also, in both the VRE and the biological growth environment, it is frequently impossible to construct an acceptable quality isosurface model that, at the same time, consists of sufficiently few triangles that they can be drawn at high frame rates. As a result, users are often confronted with high quality isosurface models with which interaction is almost impossible due to low frame rates, or with low quality models that can be displayed at acceptable frame rates. Most of these problems are the direct result of graphics and computing hardware limitations. Given time, the increase in performance and capabilities of future hardware may allow at least some of these problems to be overcome. However, as already noted in chapter 1 of this thesis, an increase in computing performance is invariably accompanied with an increase in problem size so that hardware limitations will always remain a problem.

Of all the experiences our users had with the environments, what had the most impact was the acuteness of the immersive experience in the virtual environments. Almost instantly, the users of our environments were impressed with the fidelity of the virtual objects floating in front them, which in some cases even made them want to reach out and touch the virtual objects. This quality can be mostly attributed to the combination of the surround-screen projection system of the CAVE, the stereoscopic images and head motion parallax. Our environments have also been used in other research projects, which in some cases resulted in the discovery of artifacts that had not been noticed before. For example, the VRE environment has been used for the visualization of 3D confocal laser scanning microscopy (CLSM) data from the molecular cell biology research group of the University of Amsterdam where they found spatial configurations in the nucleus of biological cells that had not been detected before.

The biggest problem for most inexperienced users was the interaction with the environments. In relatively simple environments (such as the car crash simulation playback environment), the number of functions offered by the application are sufficiently few that users have little difficulty interacting with the environment. However, in more complex environments, it has proven to be difficult to provide the environment's functionality in an intuitive manner. For one, this can be attributed to the lack of graphical user interface (GUI) toolkits for virtual environments. These toolkits would enable the application developer to extend the interaction through hardware devices (like the wand, buttons and joystick) to graphical interaction techniques (such as graphical buttons, sliders and menus) to create user interfaces that the user knows well from desktop applications. In chapter 4 we describe several techniques that address interaction in virtual environments.

All the environments described in this chapter have in common that the explored data is static or time-invariant; the data has been collected at a specific point in time; after data acquisition or after a simulation has finished. Chapter 5 addresses the

issues involved in the construction of dynamic environments that allow a researcher to explore the progress of computational processes while they are running.

Chapter 3

The UvA-DRIVE virtual reality system*

"The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, a bullet displayed in such a room will be fatal. With the appropriate programming, such a display could literally be the Wonderland into which Alice walked."

Ivan Sutherland, The Ultimate Display, Proceedings of the IFIP Congress, 2*, pages 506–508, 1965.*

3.1 Introduction

The objective of a Virtual Environment (VE, the environment that is generated by a VR system) in general is to fool a human being into believing that he or she is physically located in a synthetically generated environment by presenting him or her with reactive external stimuli. Of the human sensory system (vision, hearing, smell, touch and taste), the modality that is most often exploited in a VR system is vision because it has the greatest and most immediate impact. High-end VR systems, like the CAVE installed at SARA in 1997, have been successfully applied in numerous fields, ranging from architecture and engineering, to biotechnology, psychology and medicine [5, 20, 25, 50, 51].
What all these projects have in common is that they need a VR system that can generate a VE that is so compelling that the user is convinced he is immersed into a new world. To achieve this, the VR system must, apart from presenting the user with sensory stimuli, be able to sense the behaviour of a user so that it can react to

*Parts of this chapter have been published in *R.G. Belleman, B. Stolk, and R. de Vries. "Immersive virtual reality on commodity hardware",* Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging (ASCI 2001), pages 297–304, 2001.

the user's actions. Such an environment then allows the user to interact with the VE, thereby increasing a user's awareness of the world presented around him. It is this combination of presentation and interaction that makes VR so interesting.
Having said that, it is curious that VR has not met wider acceptance than it has up until now. The main reason for this is cost. Not until recently, the only systems that were capable of providing high fidelity VR were large monolithic computer systems with dedicated graphical subsystems that cost millions. However, with currently available commodity off-the-shelf (COTS) hardware and open source software it is now possible to build VR systems that rival and in some cases surpass the capabilities of large commercial VR systems.
This chapter describes a design that has resulted in the construction of, amongst others, the *University of Amsterdam Distributed Real-time Interactive Virtual Environment* (UvA-DRIVE), a fully functional VR system based on COTS hard- and software. Section 3.2 discusses the requirements of VR systems and the considerations that have to be taken into account while constructing one. Section 3.3 describes a number of design options for a VR architecture. Section 3.4 presents the prototype which has been built and the results of a comparative benchmark. Finally, in section 3.5 we present our conclusions.

3.2 Requirements and considerations

The primary objective of this work is to design and build a fully functional VR system that allows users to explore their datasets or to develop prototypes of virtual environments for later use in larger systems, such as a CAVE, without having to use these more expensive resources during development. The considerations we made in the design of the new architecture are the following.

Multi user
In most applications, a researcher wants to explore and discuss the results of an experiment with his peers. The display system should therefore allow more than one person to join in the experience. This rules out the use of head-mounted displays (HMD) that essentially provide a single user VR experience (the increase in both complexity and cost make it impractical for every user to wear an HMD). Furthermore, conventional cathode-ray tubes (CRT) monitors are too limited in size; in practice, no CRT monitors larger than approximately 100 cm in diagonal exist. Larger sized LCD panels are available but suffer from low refresh rates which makes them unsuitable for VR displays.

Minimal effort in porting existing software
In the past, extensive effort has been put in the development of visualization and VR applications for use on high-end VR systems. An important requirement is, therefore, that the use of this software can be continued on the new architecture with minimal porting efforts.

Processor and graphics performance
The data sets that are currently explored by the users of VR systems range anywhere from low to high volume and complexity. To accommodate the full range, the architecture's processing power and graphical performance should be able to handle these datasets adequately.

Stereoscopic 3D display
Stereo vision adds a compelling depth-cue over perspective projection by providing the left and right eye with slightly different images, corresponding to the differences the eyes see when looking at objects in the real world. This property, together with the ability to interact with a VE is what makes a VR system a unique tool for exploring large, complex, multi-dimensional data sets.

Head and hand tracking
Head tracking makes an additional depth-cue possible known as "motion parallax". With this technique, the environment responds naturally to the movements of the user so that one can attain additional information on the shape and size of a 3D structure. Hand tracking allows a VR system to determine the position and orientation of the user's hand so that interaction with the environment is possible.

Flexibility and scalability
Rapid advances in the semiconductor industry make computing and graphics systems virtually obsolete within a time span of three to five years. To lengthen this time span, the new architecture should consist of components that can be easily replaced as better hardware becomes available.

Cost
For most end-users, project budgets are limited. The solution we are looking for therefore aims at providing a VR architecture that is low cost (both in construction and maintenance) but not at the expense of the requirements mentioned above.

3.3 Design options for a VR architecture

Given the requirements and considerations outlined in the previous section, we have investigated a number of options to construct a VR architecture based on commodity hardware. During our investigation we have considered a number of different designs which will be discussed in this section. The architecture we have selected and built is described in section 3.4.

3.3.1 Multi user stereoscopic display

The display system is that part of the VR system the experimenter looks at and interacts with. In some cases, a quality monitor may be quite sufficient as a display system but for multi user use, a monitor is often too small. A convenient method for obtaining a large display area is to use high brightness projectors that project images on large surfaces that can be viewed by multiple users at the same time. In these systems a projector projects images on a screen, mostly on the back so that the user can move freely in front of the screen without occluding the projected images (see Figure 3.1). This chapter will only cover single screen projection systems.

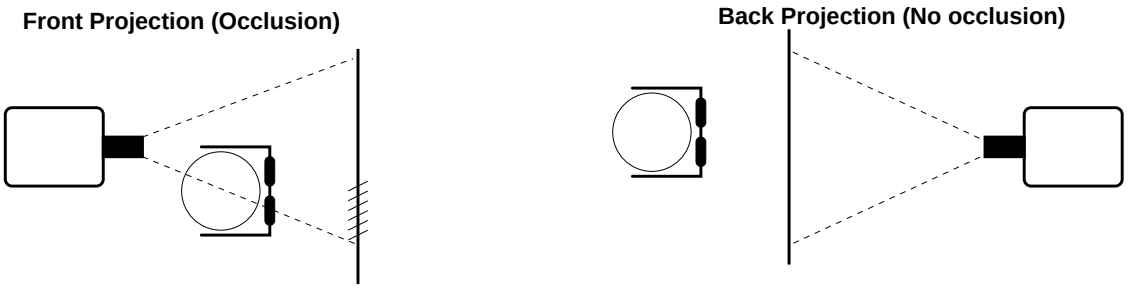


Figure 3.1: Front and back projection systems.

For projection based systems, there are basically two methods to generate stereoscopic images; active and passive stereo systems.

Active stereo

In active stereo systems, a single display device (i.e. a graphics adapter and a projector) is used to generate images for the left and right eye alternately. The display of the images is synchronized with a device that ensures that the users see only the left image in the left eye and the right image in the right eye.
The application, the graphics adapter's firmware and the graphics adapter's hardware must support this type of stereo through an interface that signals the end of a frame. This is commonly done using the vertical retrace signal that is generated by a graphics adapter at the end of a frame. To prevent eye fatigue, the frame rate of an active stereo display should be at least 100 Hz (i.e. 50 Hz per eye).
Active stereo systems either use shutter glasses or active polarization filters to direct the left and right eye images into the correct eye (as will be illustrated using Figures 3.2 and 3.3).

Active stereo using shutter glasses

Shutter glasses (see Figure 3.2) use a liquid crystal material that can be turned opaque or transparent under hardware or software control. The glasses are controlled by the graphics system, either through a wire connection or via wireless infra-red, in sync with the rendered left/right images. Although this method of generating stereo images is the most commonly used, the shutter glasses can be quite expensive and

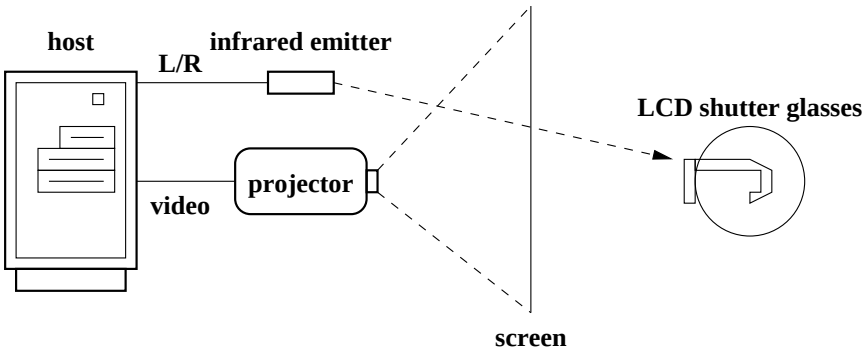


Figure 3.2: Active stereo using liquid crystal shutter glasses.

often get uncomfortable over long periods of use.

Active stereo using a "Z" screen

A "Z" screen (see Figure 3.3) is a dynamic optical filter that alternately changes the polarization direction of light that travels through under hardware or software control. The low cost, lightweight glasses worn by the users also contain polarized material that only passes light with a specific polarization direction. Linear polarization is used ("left-right" and "up-down"), which implies that the users should keep their head aligned with the Z screen in order to avoid that images for one eye "bleed through" to the other. Circular polarization filters do not have this problem but these only exist for passive setups.

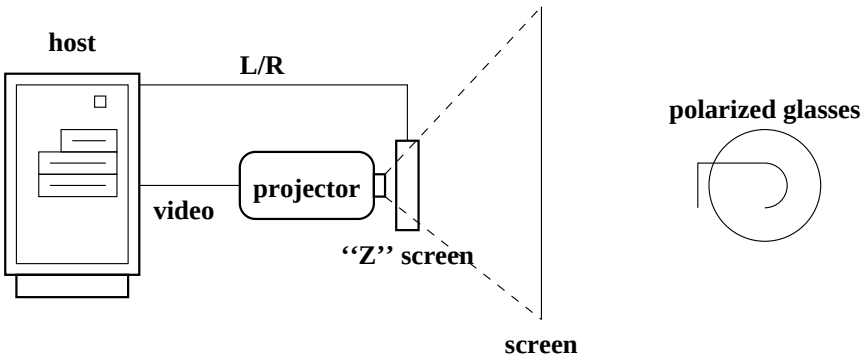


Figure 3.3: Active stereo using an active polarization screen ("Z" screen) and polarized glasses.

Passive stereo

In passive stereo systems, two display devices are used; one for each eye (see Figure 3.4). Static polarization filters are used to polarize the left and right eye images from the projectors while the users wear low cost, lightweight polarized glasses to direct the left/right images into the correct eye. Again, linear polarization is used in most

cases so that the users must keep their head aligned with the polarization filters to avoid image bleed. Circular polarization filters exist that do not exhibit this problem but these are very expensive and adversely influence image brightness.

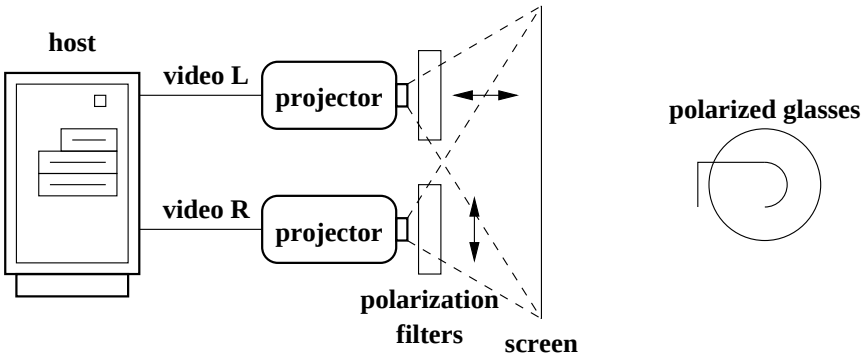


Figure 3.4: Passive stereo using two projectors, polarization filters and polarized glasses.

This solution requires "dual-headed" support by both the software and the hardware[†] as *two* graphics adapters are used. Another disadvantage with using two projectors is that the projectors need to be accurately aligned. With a single projector, calibration is considerably easier.
In any polarization system, special care has to be taken with the choice of screen material since most projection screens adversely influence the polarization direction of incident light. This is especially the case with the use of back-projection systems and even more so with circular polarization.

3.3.2 High performance 3D graphics adapters

Graphics hardware performance has increased dramatically over the last years. Although this hardware was mainly intended for use in games, the capabilities of these adapters make them very well suitable for scientific visualization and in some cases rival the capabilities of commercial solutions. With the introduction of 3D accelerated chipsets that include hardware support for 3D operations such as linear transformations, lighting and depth buffering, powerful graphics hardware is now within reach of everyday consumers.

OpenGL

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications [261]. With its low-level software interface, OpenGL has often been called the "assembler language" of computer graphics. Applications in many domains, including entertainment, manufacturing, medical imaging and more, have

[†]Note that most graphics adapters for PCs use the AGP interface while no PC motherboards existed at the time of writing which have more than one AGP slot.

benefited from OpenGL's multi-platform accessibility and depth of functionality. Since SGI introduced OpenGL, it has grown into the leading cross-platform graphics Application Programmer's Interface (API).

Hardware acceleration

OpenGL's design lends itself very well for hardware acceleration. Hardware accelerated graphics adapters have been available for SGI systems since the early days of OpenGL. But mostly due to the advances in the gaming industry, semi-conductor manufacturers are now rapidly closing this historical gap and in some cases surpass the performance of SGI's hardware.
For most high-end graphics adapters, hardware accelerated OpenGL support for various operating systems is available either directly from the vendor or through third-parties. However, this support is in some cases experimental and lacks features that are desired for VR purposes (such as stereo support).

Stereoscopic 3D

The essential part of stereoscopic rendering is the generation of two video streams; one for the left eye and one for the right eye. The task of generating and managing stereo pair video streams should not be the responsibility of the application developer. Instead, this should be fully transparent to the developer.
The common approach taken by most graphics adapter manufacturers is to use four frame buffers; images for the left and right eye are drawn into a "back" buffer while the user looks at the left and right images in the "front" buffer (commonly referred to as "quad buffered stereo"). While most 3D graphics adapters support quad buffered stereo in hardware, this support is not always reflected in the driver software for some operating systems. In cases where no quad buffered stereo support is available, another method must be used to generate stereo pairs. For the display options described in section 3.3.1, there are a number of alternatives.

Stereoscopic 3D using X Windows

The *de facto* standard graphics system for Unix environments is the X Windows system. X Windows uses the concepts of hosts, displays, screens and windows (arranged from broad to narrow scope, see Figure 3.5). X Windows is able to access any graphical display on a connected system using an identifier of the form `hostname:displaynumber.screennumber`.
Using X Windows, we can opt for one of the following configurations to generate stereo image pairs:

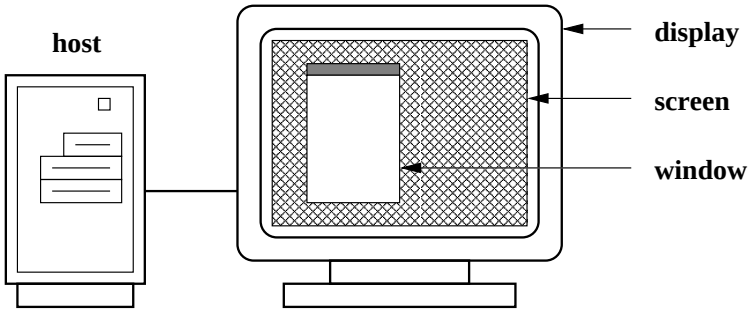
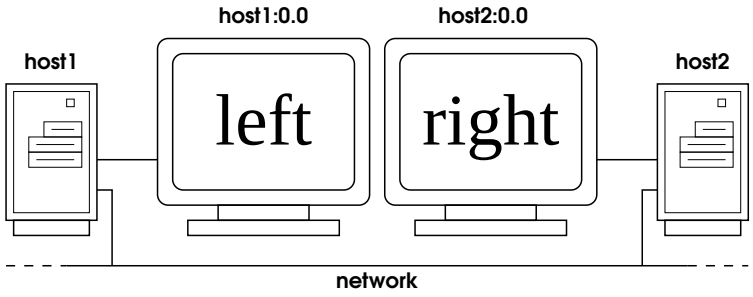


Figure 3.5: The X Window system concept of hosts, displays, screens and windows.

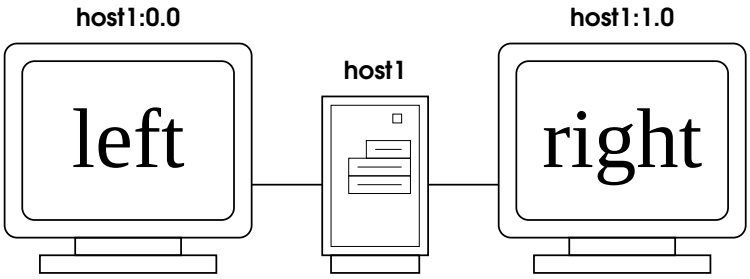
Two hosts, two graphics adapters:



Here two hosts are connected via a network, each equipped with a graphics adapter and display device (a monitor or projector). Using a display identifier as described earlier, each host can access the other host's display[‡]. This configuration is suitable for use in a passive projection system (see Figure 3.4).
The greatest advantage of this configuration is that left and right eye images can be rendered by each graphics adapter in parallel, which *can* improve performance. However, in practice, one of the two hosts will run the VE application which computes the view for *both* the left *and* right eye. Then, it renders its own view on its local adapter and the other view is sent over the network. The other host then only has to render the graphics that it receives over the network. This implies a load imbalance which may result in synchronization problems if no countermeasures are taken.
A possible solution to these synchronization problems would be to run the application on a third system, but in that case network traffic doubles since now rendering information for *both* eyes must be communicated.

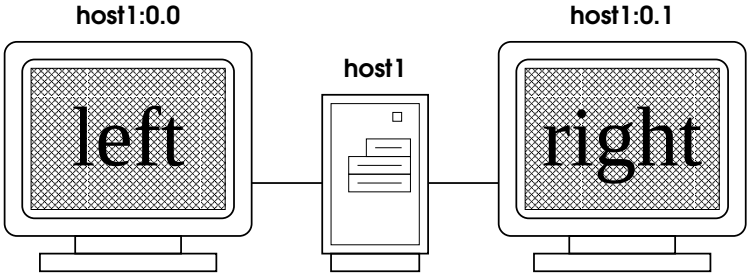
[‡]The X Window implementation should support the GLX protocol in order to run OpenGL applications over a network.

One host, two graphics adapters:



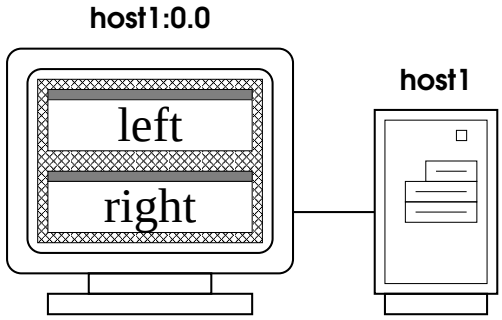
In principle, the synchronization problems described above would be solved by using two graphics adapters in one host (parallelism is then sacrificed, of course). In that case the application renders the left/right views to two displays that reside on the same host. However, this configuration can not be built from commodity hardware since (as noted earlier) most graphics adapters use the AGP interface while PC motherboards only have one AGP slot. Some graphics adapters exist for a PC's PCI interface, allowing an additional card to be added, but it should be noted that the performance of these cards is often substantially less than that of AGP based cards (mainly because of the lower bandwidth of the PCI bus), again resulting in synchronization problems.

One host, multi-headed graphics adapter:



Multiple screens on the same host can be achieved by using a graphics adapter with multi-head output. Synchronization can be controlled since the two views are computed on one host and rendered to two outputs on the same graphics adapter.
Care should be taken that the operating and graphics system include multi-headed support (for example; XFree86, an X window implementation available for various operating systems, supports multi-head since version 4.0.1, released late 1999, but it does *not* support accelerated OpenGL to both heads).

One host, one graphics adapter and sync-doubler:



Specialized hardware in the form of a "sync-doubler" can be incorporated in cases where none of the alternatives described above can be applied. A sync-doubler is a device that accepts as input a video signal of X Hz, and outputs a signal of $2X$ Hz, whereby the frames are constructed by using the top and bottom half of the input image in an alternating fashion. The top and bottom halves often need to be separated by a number of blank lines.
This simplifies the task of generating a stream of mixed video frames to generating a video stream composed of top/bottom halves. A drawback of the sync-doubler method is that vertical resolution is halved.

3.3.3 Tracking and input devices

Position and orientation are typically determined through six degrees-of-freedom (DOF) tracking sensors (or "trackers"). A tracker mounted on the user's head allows a VR system to provide user centred projections that correspond to a user's displacements ("motion parallax"). A hand tracker allows the user to interact with virtual objects. VR input devices often combine trackers with a number of buttons that allow the user to convey intention to the VE, similar to the buttons on a mouse on personal computers.
There are different types of tracking systems, each based on different acquisition techniques such as magnetic, acoustic, optical and inertial trackers. Each of these have their strong and weak points but to describe these would go far beyond the scope of this chapter. By far the most popular in VR systems are the magnetic tracking systems built by Ascension and Polhemus. Although these tracking systems have been in existence for many years, they are still very expensive due to the unavailability of acceptable alternatives. A positive side effect of this is that they are supported by nearly all VR application development environments.

3.3.4 Software availability

Since most of our previously developed software runs on Unix operating systems and makes use of OpenGL, SGI's OpenGL|Performer, VRCO's CAVE library, Kitware's

Visualization Toolkit and DMSO's High Level Architecture, a prerequisite is that this software is available for the new architecture.
OpenGL has already been described in section 3.3.2. The following provides a brief overview on the other software packages.

OpenGL|Performer

OpenGL|Performer is a programming interface for creating real-time visual simulation and other performance-oriented 3D graphics applications [190]. It simplifies the development of VR applications through a low level library providing rendering functions, a scene graph and rendering system, functions for defining both geometric and appearance attributes of three dimensional objects, user-interface components and support for many popular industry standard database formats. OpenGL|Performer is currently available for IRIX and Linux systems. Microsoft Windows support will be added in version 3.0 which is scheduled for release in late 2002.

The CAVE library (CAVELib)

CAVELib is an Application Programmers Interface (API) that provides general support for building virtual environments [249]. CAVELib configures display devices, synchronizes processes, draws stereoscopic views, creates a viewer-centred perspective and provides basic networking between remote Virtual Environments. A flexible configuration method makes programs written with CAVELib portable to a widevariety of display and input devices without rewriting or recompiling. CAVELib currently supports most common Unix systems. Microsoft Windows support was added in 2002.

The Visualization Toolkit (VTK)

VTK is an open source, freely available software system for 3D computer graphics, image processing, and visualization [209]. The design and implementation of this library has been strongly influenced by object-oriented principles. VTK includes a C++ class library containing over 500 visualization objects, and several interpreted interface layers including Tcl, Java, and Python. VTK runs on nearly every Unix based platform and Microsoft Windows.

The High Level Architecture (HLA)

The High Level Architecture (HLA) aims to establish a common architecture for simulation to facilitate interoperability among simulations and promote the reuse of simulations and their components [165]. As a successor to the DIS (Distributed Interactive Simulation) protocol, HLA provides a robust architecture with which distributed discrete event and other types of simulations can be designed. One particular implementation of HLA, the Run Time Infrastructure (RTI) developed by the Defense

Modeling and Simulation Office (DMSO, United States Department of Defense), is supported on all major computing platforms.

3.3.5 Computing hardware

The computing hardware performs the main processing in the system, delegating hardware specific tasks to dedicated subprocessors. For a high performance VR system, the components used in the computing hardware should be selected with some care. For example; as most VR applications benefit from a multi processing design, the performance of the system as a whole can be increased by applying a multi-processor design. The easiest multi-processor solution would be a Symmetric MultiProcessing (SMP) architecture in which all processors share the same resources in the system. Although the maximum number of processors in an SMP architecture is limited, it makes porting of existing software far easier when compared to distributed multi-processor systems where some form of synchronization will need to take place.

Because graphical performance is important in VR applications, the communication bus between the main CPU and the graphics adapter should be of sufficient bandwidth to handle all data communication within the time constraints imposed by the application. One particular example of a bus architecture in which the communication bandwidth between the main processor and the graphics adapter is optimized can be found in most personal computers (PCs), called the Accelerated Graphics Port (AGP). The initial implementation of AGP allowed for a peak bandwidth of 264 MB/s which was almost double the 133 MB/s bandwidth provided by the main PCI bus found in PCs at that time. In later versions of AGP, bandwidth was increased further by allowing multiple data transfers per clock cycle which resulted in bandwidths of 528 MB/s (for AGP 2×), 1056 MB/s (for AGP 4×) and 2112 MB/s (for AGP 8×).

Most manufacturers of computing hardware have taken some, or all of these issues into account in their designs. Mass acceptance has caused PC technology to be able to rival, both in terms of performance and quality, that of large commercial computer hardware manufacturers but at a significantly lower price. In addition, PC based hardware allows high flexibility and scalability in the sense that hardware can be replaced by newer (in general; faster, better) hardware easily and at low cost as soon as it is available.

Operating system

Several operating systems are available for Intel processor based PC platforms, including IRIX, Solaris, various flavours of BSD and Linux. However, not all of these provide both the hardware and software support required for compatibility with the VR applications as described in section 3.3.4. The Linux operating system in this respect provides the most complete support.

3.4 The design of UvA-DRIVE

After careful consideration of the requirements posed in section 3.2 and the design options outlined in section 3.3, we have opted for an Intel PC based computing system in conjunction with an active shutter glasses system as shown in Figure 3.2. The operating system running on this system is Linux. A first prototype of our design was built by SARA in 2001, called the Linux Immersive Environment (SARA LIE). Multiple systems have been derived from this prototype, including UvA-DRIVE.
Unfortunately, at the time of construction, none of the drivers for OpenGL accelerated adapters that were supported under Linux contained quad buffered stereo support[§]. As an intermediate solution for the prototype, we use the "sync-doubler" technique described in section 3.3.2 to generate stereo pair video streams. The sync-doubler also drives an infrared transmitter that switches the shutter glasses. As we have opted to use an active stereo approach, the system requires only one projector. A schematic representation of the architecture is shown in Figure 3.6. Table 3.1 provides a list of components and specifications of both SARA LIE and UvA-DRIVE compared to the SARA CAVE.

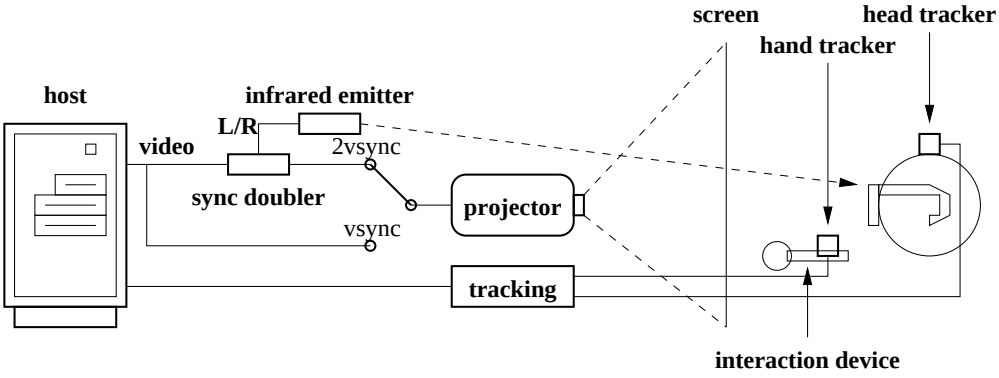


Figure 3.6: General setup of the VR architecture: an active stereo projection system using sync-doubling technology and an interaction system based on tracking hardware.

3.4.1 Top-bottom stereo with CAVELib

Since all previously developed VR software is based on VRCO's CAVELib, we have chosen to use CAVELib to minimize porting effort. Using CAVELib, the application programmer can ignore technicalities like the number of projection screens, the setup of the projection screens, the mono/stereo properties of projection screens. Instead, the application developer can focus on that what matters: the 3D scene. How this 3D scene is conveyed to the user is irrelevant from the developer's point of view.
CAVELib has a feature that allows the rendering for left/right eye to be performed at a specified sub window of the physical screen. Also, CAVELib supports applications

[§]*nVidia* (a manufacturer of high performance graphics chips used in many PCs today) released Linux drivers with quad buffered stereo support on September 10, 2002 (version 1.0-3123) [173].

	SARA CAVE (1997)	**SARA LIE** (2001)	**UvA-DRIVE** (2002)
CPU	8 × MIPS R10000	2 × Intel Pentium-III	2 × Intel Pentium-III
CPU clock	195 MHz	700 MHz	1.0 GHz
architecture	Onyx2 Reality Monster	ASUS P2B-D, SMP Intel 440BX chipset	ASUS CU4VX-D, SMP VIA Apollo Pro chipset
memory	1 GB	256 MB, 100 MHz FSB	1 GB, 133 MHz FSB
graphics	4 × InfiniteReality2	Creative Labs nVidia GeForce2 GTS	ASUS V7700 (modified) nVidia Quadro2 Pro
gfx memory	4 × 16 MB	32 MB video memory	64 MB video memory
CPU→gfx bus	800 MB/s	528 MB/s (AGP 2×)	528 MB/s (AGP 2×)
projector	Electrohome Marquee 8500	Electrohome Marquee 8500	Electrohome Marquee 8110+
screens	4 (CAVE)	1 (IDesk)	1 (ImfinityBox)
tracking	Ascension FoB four sensors	Ascension FoB two sensors	Polhemus Fastrak two sensors
OS	IRIX 6.5	Linux Debian (“sid”) kernel 2.2.18	Linux RedHat 7.3 kernel 2.4.19
software	X Windows OpenGL 1.1 Performer 2.3 CAVELib 2.7	XFree86 4.0.2 nVidia GLX 0.9-6 Performer 2.3 CAVELib 2.7	XFree86 4.0.1a nVidia GLX 1.0-2960 Performer 2.3 CAVELib 3.0.1

Table 3.1: Comparison of components and specifications of the CAVE, SARA LIE and UvA-DRIVE.

that use OpenGL|Performer. This creates two additional advantages: minimal porting effort for the migration from SGI (CAVE) to Linux, save a recompilation of the sources, and also good possibilities for performance measurements comparing SGI and PC based VR.
XFree86 was configured to run at 1024x1576 resolution with a colour depth of 24 bit per pixel. This means, that after sync-doubling, (1576-40)/2 = 768 pixels remain in the vertical direction[¶]; the same as on the SGI Onyx platform used in the SARA CAVE. We have found that a vertical sync of 85 Hz (170 Hz when doubled) is possible, provided that a high quality projector is used.

3.4.2 Performance measurements

We have run benchmarks with full tracking support and stereo vision support. The program to test the performance is an application developed by SARA, on top of VRCO’s CAVELib.
In benchmark A, the dataset consists of 16 animated objects. Each object consists of 9120 triangles (a total of 145920 triangles), organized in 405 triangle strips. In benchmark B, the dataset consists of a surface model extracted from an Computed

[¶]The sync-doubler requires 40 blank lines between windows.

Tomography (CT) dataset using a surface extraction algorithm. The dataset contains 204480 triangles, organized in 39134 triangle strips.
Two platforms were used for benchmarking. The first is SARA's CAVE facility; an SGI Onyx2 Reality Monster with 8 R10000 CPUs at 195 MHz, running IRIX 6.5. This system is equipped with 4 InfiniteReality2 graphics pipes. For benchmarking, we disabled all but one screen. The CAVE ran the benchmarks at a stereo 1024x768 resolution, 60 Hz refresh rate (120 Hz stereo), and full screen anti-aliasing.
The second system is the SARA LIE system described in section 3.4, and ran the benchmarks at a stereo 1024x768 resolution, 58.4 Hz refresh rate (116.8 Hz stereo), and without full screen anti-aliasing[‖]. Both tests did not include texture mapping.

	test A	test B
CAVE	13.3	5.5
SARA LIE	11.7	6.3

Table 3.2: Performance in frames per second of the SARA CAVE and the SARA LIE prototype in two tests.

The results (see Table 3.2) show that there is no significant performance difference between the two platforms. The PC based system performs better on test B. This is mainly due to the fact that a single Intel CPU has a better floating point performance than a single MIPS CPU [181].

Figure 3.7: The UvA-DRIVE system.

3.5 Conclusions

Immersive VR on commodity hardware shows great promise. Although not all the required hard- and software components for a single screen immersive VR system based

[‖]Although the nVidia GLX implementation supports anti-aliasing, we could not get CAVELib for Linux to enable this mode.

on commodity-of-the-shelf are available yet, progress is made or satisfactory alternatives exist. Our first experiences show that a PC based system is a viable alternative to established VR systems. We have shown that graphics performance for PC based solutions is in the same order as that of high-end VR systems and is therefore no longer a criterion for choosing one solution over the other. Moreover, compatibility with existing VR applications is provided with currently existing solutions. Because PC based VR systems are affordable, offer adequate performance and are compatible with high-end VR systems, a host of new application domains now comes in reach.
Currently, work is in progress at the Section Computational Science for the construction of a PC based near-field VR system called the "Personal Space Station" (PSS). This design by Mulder et al. uses a conventional computer display in combination with a mirror and a camera based tracking system [168] (see also [213] and [186]). As this design does not use a projector or a large projection screen, it can be built at significantly lower cost and is suitable for personal use in a normal office environment. A PSS is well suited for applications in which direct, hand-eye coordinated interaction with virtual objects is important.

Chapter 4

Enabling technology for interaction in virtual environments*

"People shouldn't have to read a manual to open a door, even if it is only one word long (push/pull)."

Donald A. Norman.

4.1 Introduction

Virtual Environments (VEs) are used in areas where the immersion of a person in a synthetically generated environment provides more insight in a specific problem over classic methods using a desktop [247]. Often, these type of problems entail the analysis of three or more dimensional structures that are difficult to comprehend using projections on two dimensional screens. The primary objective of an immersive VE is to engulf a user in a computer generated synthetic environment. A successful immersive experience is achieved when the user experiences the sensation of "presence", of being part of the artificial environment. Interaction is a key requirement in achieving this goal.
In this chapter we begin by identifying the interaction methods that are important to do useful work in a VEs. In the sections that follow, we describe various interaction techniques that can be used to create efficient, flexible and information rich immersive exploration environments.

4.1.1 Increasing awareness through interaction

Different sensory modalities can be used to present data to the human senses. In Virtual Reality (VR) research the objective is to subject the human sensory system with impulses that leads the user to believe he is present in the synthetic world. As

*Parts of this chapter have been published in *R.G. Belleman, J.A. Kaandorp, D. Dijkman and P.M.A. Sloot.* "GEOPROVE: Geometric Probes for Virtual Environments", number 1593 in *Lecture Notes in Computer Science*, pages 817–827, 1999.

in the real world, the experience of presence is greatly enhanced when the environment responds to our interaction. Interaction in an immersive VE should *enhance* the immersive experience and most definitely not break it through interaction methods that force the user to step out of the VE and into the real world, not even for a brief moment. Unfortunately, very few readily available toolkits exist for VE application developers that facilitate the construction of interactive VEs. The ones that *do* exist use specialized hardware or physical devices that the user wears whilst in the VE, offer limited flexibility, are too specific to the application area for which they were developed or they require a substantial software engineering effort from the developer [35, 43, 127, 149, 161, 201].

Input devices, gestures, intention and mapping

Input devices are the primary interfaces between the real and virtual world that enable users to interact with a VE. Input devices offer a degree of expressiveness that is proportional to the "degrees of freedom" (DOF) they express. A simple button represents either a pressed or unpressed state and therefore does not provide sufficient expressiveness to denote, say, a two dimensional vector like a two dimensional analog joystick does. Higher degrees of expressiveness can be achieved by combining input devices.
A frequently used input device used for interaction in a VE are 6 degrees of freedom (3 translation and 3 rotation) tracking sensors. These devices create a relation of the position and orientation of physical objects (such as the user's hand) with objects in the virtual world. Common ways in which these devices are used to interact with VEs are through proximity tests that allow virtual objects to be "grabbed" or intersection tests that detect that the device is pointing at a virtual object. Input devices used in VR systems are often a combination of a tracking sensor with "manipulators" like buttons, small joysticks or flex sensors (as in gloves) that can be manipulated by the user.
VR input devices are used to monitor the user's "gestures" which in turn convey "intention" to the environment. The user's gestures are analysed by the environment to ascertain the user's intentions which are then "mapped" to interaction methods. An easy to use VE accurately maps the user's gestures to the associated interaction method so that the environment fulfills the user's intention [162]. Moreover, the interaction methods provided to the user should provide clear and intuitive clues to their "affordance"; how they should be used and what they can be used for [171]. For example, clinching the fingers of a glove in the proximity of a virtual object or clicking a button while pointing at an object could result in the object being selected.

4.1.2 Interaction methods

Some interaction methods in a VE will be predetermined by the task-specific actions that have to be performed. However, most environments have a need for interaction methods that are generic for most applications. Unfortunately, there is no unifying

framework for interaction in VEs as there are now for desktop PCs [162, 201]. Due to its immersive nature, there are, however, a number of interaction methods that we feel are mandatory to do useful work in a VE.

Motion parallax

An important interaction method allowing a user to comprehend multi dimensional structures is by allowing the user to look around the visual representation of these structures. Just as in the real world, the difference in perspective that results from the changing position of the user's eyes in relation to the object provides information on the size and relative location of substructures, thereby providing useful depth-cues. This method of interaction is often referred to as "motion parallax". In VEs, motion parallax is achieved by changing the rendered images based on the location and orientation of the head. To that end, a tracking sensor is mounted on the head, allowing the VE to obtain this information.

Object manipulation

The content of virtual environments consists of objects that are representations of the structures of interest to the user. The manipulation of these objects can form the basis of many interaction methods, much in the way we manipulate objects in the world around us. An obvious (and intuitive) way to implement interaction in immersive VEs would, therefore, be through the manipulation of virtual objects in the environment [22,24,135,163,164,167,188]. In general, the following object manipulation tasks can be identified:

- create, delete - The instantiation or removal of an object from the environment changes the context of the environment, which in turn can be a reason to trigger an action. For example, the deletion of an object could result in the deletion of an associated menu (and vice versa). We will come back to what we mean by "context" in section 4.4 (page 81).

- select - Selection is equivalent to identifying an object as being in the focus of the user's attention. If the object is a visual representation of an action, selecting an object may result in the execution of an action (much like pressing a button).

- move, rotate, scale - Objects in a VE have a position, orientation or scale which may be changed. This change itself may represent a change in context (like moving a chess piece on a board) or represent a quantitative value (like the displacement of a button over a certain distance in a slider).

- parameterization - Besides position, orientation and scale, objects often have other, more object specific attributes associated with them like shape, or color. Interaction methods to change the value of an attribute is a useful type of interaction to signify parameter changes. The type, range and accuracy of the

attribute puts constraints on the interaction methods that are most suitable to set the value of an attribute.

In section 4.2 we describe an architecture that extends an existing scientific visualization environment for use in a VE. This architecture allows for the manipulation of scientific visualization results to facilitate the construction of information rich and flexible exploration environments for scientific research.

Navigation and wayfinding

Users are constrained by the physical confinements of the VR devices they are using. For example, in the case of HMDs the user will not be able to move beyond the length of the connected cables, in projection based systems, the projection screens limit the user's movements. The virtual environment that they explore is often larger than these confinements so that interaction methods are required that allow the user to navigate through the virtual world and reach objects outside physical reach or to constrain the movements of the user in relation to objects [132, 187, 198]. While the user is navigating through a VE, it may be hard to find one's way around and to locate "what is where" in the environment, especially in large VEs [55, 73, 74, 179].

Quantitative and text input

Text and number entry are important in situations where precision is critical. For some applications an approximate value may be sufficient so that some form of user-interface widget (such as the slider or dial we know so well from desktop user interfaces) could be used. Unfortunately, none of the existing user-interface libraries that exist today can be readily incorporated into a VE. Also, in other situations where exact values are critical, interaction methods are required that allow the user to enter individual digits or characters. Some approaches to achieve exact value input involve physical input devices that are taken into the VR system, such as keyboards, tablets and even small computers (like PDAs) [252, 259]. For projection based environments this is fine, but for head-mounted-displays this solution fails since the physical devices cannot be seen by the user. In section 4.3 we address these problems through an architecture that allows existing desktop applications and 2D user-interface libraries to be used in a VE.
Voice input is an alternative method that is suitable for exact value input in some cases [42]. Number entry is viable using most voice recognition systems through a well-defined vocabulary [107]. Text input however forms a far more difficult problem since a predefined vocabulary can, in most cases, not be defined. Although most automatic speech recognition (ASR) systems provide a dictation mode that allows free text to be entered, the accuracy of these systems in this mode leaves a lot to be desired [208, 264]. In section 4.4.2 we describe a speech recognition technique for quantitative and text input that exploits environment context to increase recognition reliability and allows multimodal interaction in a VE.

Quantification

Although stereopsis allows humans to perceive depth, it is know that distances and sizes in virtual environments are frequently underestimated due to a lack of adequate visual stimuli for scale [200]. An instrument for obtaining quantitative information from a visual representation is therefore a valuable asset. Section 4.5 describes GEOPROVE, a geometric probing software architecture for interactive data exploration environments, virtual environments in particular. This architecture allows researchers to probe visual presentations in order to obtain quantitative information. The properties that we want to measure could be obtained automatically using data analysis techniques, but this often requires designing and implementing specialized algorithms that are dedicated to the specific task. Quite often these techniques rely on heuristic algorithms that are difficult to design, implement and control.

4.2 Interactive scientific visualization in VEs

The primary method used to create virtual environments is through the rendering of visual constructs. Programming environments exist that are ideally targeted towards the construction of "content" for a VE, but in general these environments do not provide methods to incorporate interactive scientific visualization into the VE [212,218,219,236]. Likewise, many desktop environments exist that provide flexible methods to do scientific visualization, but few of these provide support for direct interaction and manipulation of graphical objects in VEs [103,172,209,242].
A combination of scientific visualization methods and interaction methods that allow a user to manipulate these visualizations from within a VE would allow for the construction of interactive exploration environments for use in scientific research. Using such a combination, environments could be constructed where scientific data is represented through well established visualization methods and the interaction capabilities support exploration.

4.2.1 The Visualization Toolkit

For our research we have chosen the Visualization Toolkit (Vtk) for our scientific visualization purposes [106,209]. Vtk is an open source object-oriented software system for 3D computer graphics, image processing and scientific visualization. The availability of the source code to Vtk allows us to extend its functionality so that it can be used for interactive scientific visualization in VEs. In addition, Vtk is supported on most major operating systems which greatly facilitates application development and portability. The combination of Vtk and the CAVE library produces an extremely versatile and powerful software environment for scientific visualization in VEs.
The Vtk pipeline is an extension to the pipeline described in section 1.3.2 (page 7) and is illustrated in Figure 4.1. The third column of this Figure also illustrates how consecutive stages in the pipelines are created, parameterized and connected in a program, in this case in the Tcl language [175,258]. Vtk pipelines start with source

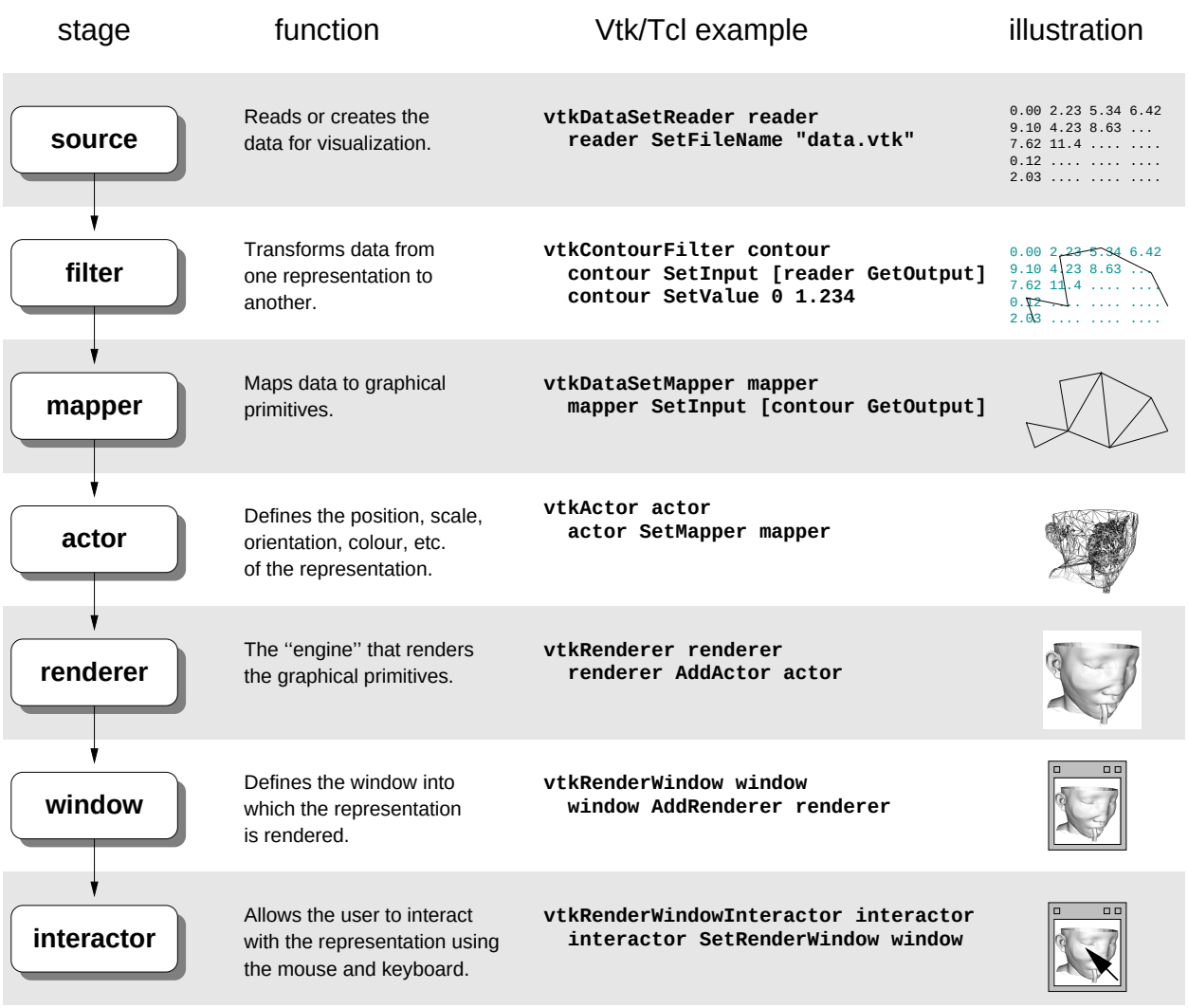


Figure 4.1: The Visualization Toolkit (Vtk) pipeline.

objects that either produce source data themselves or that read input data from external sources such as simulations or CAD programs. The output of a source object goes through several filters that alter the geometry. In most cases it is these filters that are the embodiment of a visualization algorithm. A mapper is used to map the output data from the filters to geometric constructs that can be rendered by a graphics rendering pipeline. The actor object is used to represent the resulting visualization and is used to define the type and properties of the representation. The renderer forms the interface between the visualization and the graphics rendering pipeline. On the desktop, the renderer displays the resulting images in a window while interaction with the visualization using the mouse and keyboard is implemented through an interactor. Other objects types in Vtk define light sources, cameras and mathematical functions.

4.2.2 SCAVI: Speech, CAVE and Vtk Interaction

Previous work by others resulted in a library to render Vtk objects in a CAVELib application [94]. This library copies all Vtk actors into a shared memory area to make them accessible to all display processes for rendering (please refer to Appendix A for details on CAVELib). Dynamically changing data is handled automatically. Not all of Vtk's functionality is supported (most notably cameras and lights are unsupported); we have made several extensions to the original version to support actor opacity, actor removal, visibility and texture mapping of actors. Although this work allows us to render Vtk visualizations in CAVELib applications, it did not provide methods to interact with them. We have developed a framework that allows rendering and interaction with scientific visualizations produced by Vtk. This framework, called "Speech, CAVE and Vtk Interaction" (SCAVI), provides the VE application developer with functionality to interact directly with the visualized objects using the CAVE wand and buttons and speech commands [95].

Direct object manipulation

Direct object interaction in SCAVI is implemented using a hierarchical intersection scheme: For each actor defined in the renderer, SCAVI first checks whether the line that starts at the location of the front of the wand and points in the direction defined by the wand's orientation intersects the bounding-box of the actor. If one or more intersections are found, a second line intersection test is performed on these objects and the distance from the wand to the intersection points (if any) are calculated. The object with the closest distance to the wand is considered to be "in focus". Only actors that are defined as "visible" and "pickable" can become focused (triggering a focus event) whenever there is a direct line-of-sight from the front of the wand to that object. Pressing and holding the first wand button allows a focused object to be grabbed and dragged to another location. Pressing the second wand button on a focused object selects that object, such that more elaborate manipulation can be performed. Only one actor can be selected at a time. Manipulations on actors that are in focus (by pointing at them) take precedence over actors that are selected. The application developer can request a reference to a selected actor or the actor that is in focus for subsequent action. Figure 4.2 illustrates a simple application that allows primitive objects (spheres, cubes, cones, etc.) to be manipulated interactively.
Every object in SCAVI is a Vtk actor with several enhancements defined in a C++ class called `vtkCAVEActor`. All SCAVI objects are kept in a dynamically linked list (DLL), which is not kept in shared memory making it only available to the main CAVE process and none of the display processes. The main extra features of a SCAVI object over a Vtk actor is the ability to add several event handlers to an object and the ability of the interaction component to identify and perform interaction on an object. Each SCAVI object can be given a name so that voice identification using speech technology or enumeration in a virtual menu becomes possible. Besides regular Vtk functionality, such as changing the color of the object or the visibility state, several functions have been rewritten to accommodate for the extended functionality, such as

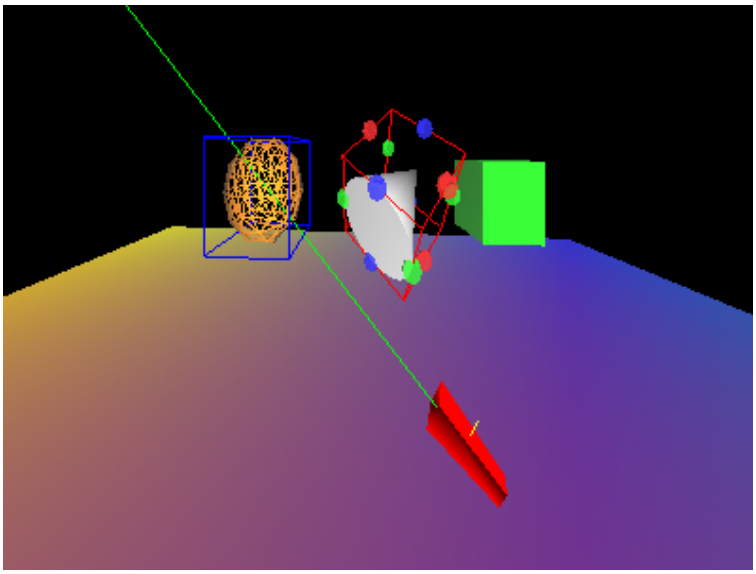


Figure 4.2: SCAVI *in use to manipulate Vtk actors.*

delete and copy. The rotating, scaling and translation functions are also rewritten, which makes it possible to transparently use either the transformation matrix at the end of the Vtk pipeline (i.e. from the actor) or after the source input. The benefit of using a transformation matrix directly after the source input is that use of subsequent Vtk filters will be aware of changes in the object's position, scale and orientation.

Although the concept of a "scene graph" is not explicitly implemented in Vtk (and most other scientific visualization environments), this functionality can be simulated through the use of a coordinate transformation filter (such as the Vtk class `vtkTransformPolyDataFilter`). This filter accepts vertex coordinates and a transformation matrix as input and produces transformed coordinates as output.

The transformation of coordinates requires a matrix-vector product for each coordinate and so the time required to transform larger objects gets increasingly large. This poses a problem in particular when a user manipulates the position, rotation or scale of an object. In a "real" scene graph system, this displacement is easily performed by inserting a transformation between the object and its parent that reflects the displacement of the hand from the moment the object was grabbed. With the use of a transformation filter, the new position of all polygons in the object must be recalculated at each frame by a transformation filter. This method may yield acceptable results for small objects, but for large objects the time spent by the transformation filter will make fluent interaction impossible.

In SCAVI this problem is circumvented by the following trick; as the user manipulates an object, a transformation matrix is associated with the actor representing the object so that the visual appearance of the object mimics the displacement intended by the user. The transformation matrix associated with the actor is used by the graphics rendering library (OpenGL) which performs the transformation in hardware, on hardware that supports it, and is therefore much quicker compared to the software

solution. As soon as the user releases the object, the actor's transformation matrix is copied and given to a transformation filter which then applies the transformation to all vertices in the object. During the interaction, the visual appearance of the object thus transforms, following the user's gestures, while the actual transformation of coordinates takes place just once when the user releases the object. As a result, there will be a slight delay from the moment the user releases the object and the time the object is actually transformed. Within this time, no interaction with the object is possible.

4.3 Bridging the gap: 2D applications in VEs

Many applications in everyday use are designed for desktop systems with 2D graphical user interfaces (GUIs). While these applications may not always be of particular use in VEs, some provide capabilities that can be very useful to the VE user or application developer. Web browsers, document readers and movie players, for example, are often used by desktop applications as "standard" utilities to provide on-line documentation or tutorials to the user. Desktop users also benefit greatly from "productivity applications" like calculators, file browsers or conferencing tools, to name but a few, that provide a rich set of accessories that enable the user to perform diverse tasks that are not offered within a single application. The same is true for VE applications; much of the functionality offered by the desktop utilities just mentioned would be quite useful to users immersed in a VE. Unfortunately, few of these utilities currently have 3D counterparts for use in immersive VEs.

A solution to this is to bring a desktop PC system or a handheld computer like a Personal Digital Assistant (PDA) into the VR installation to gain access to these utilities [252, 259]. However, this solution is impractical in combination with head-mounted displays (HMDs) since the display obscures the physical devices from view. But also in projection based VR systems, the user needs to "escape" from the VE to access the system in the real world. Others have looked at solutions in which the application is rewritten, or more specifically; the part that builds the GUI, so that the GUI is presented in the VE [6, 7]. This either requires that the utility is available in source form, and the VE application developer is up to this task, or that 3D equivalents of 2D GUI toolkits are available [43]. Few GUI toolkits for use in VEs exist, however, and the ones that do, often do not offer the same richness that is needed to effectively express the same functionality. Moreover, applications that are distributed in binary form only can not be adapted at all.

Although one can argue whether 2D interaction metaphors are efficient for use in immersive environments, the advantages are multitude. Users are accustomed to applications with 2D GUIs, so presenting them with existing utilities that can be used in a VE provides the user with familiar applications that he knows to use well. The immersive experience of the VE is maintained as the user is no longer forced to step into the real world to interact with a GUI on a desktop. VE application developers would be able to exploit existing applications, as do desktop applications developers,

and focus on the task for which the VE is intended. Application developers would be able to quickly construct GUIs using the flexibility of existing 2D GUI toolkits instead of implementing 3D equivalents.

4.3.1 XiVE: X in Virtual Environments

X in Virtual Environments (XiVE, pronounced "zive") provides access to existing 2D desktop applications by "swallowing" their GUI into the VE. The ideas behind XiVE have earlier been described by others.
Dykstra in [70] describes a modified X server that renders images into memory instead of to a graphics device and accepts device input events through a FIFO queue. The VE application then maps the images rendered by the X server onto textures in 3D space while interaction with the applications is made possibly using the FIFO queue. Developed at Chalmers Medialab, 3Dwm uses the Virtual Network Computing (VNC, [38]) remote display system protocol to distribute a bitmapped desktop across a network [71, 159]. XiVE uses a more generic and flexible approach by exploiting the client/server protocol that is used in the X Window System. The X Window System (from hereon referred to simply as "X") was designed with network transparency in mind. This means that an application (the "client") can run on one host and open windows on a "server" running on any accessible host (meaning: reachable across a network and having permission to access the server's resources).

4.3.2 Design of XiVE

XiVE maintains a list of windows that are to be presented in the virtual environment (the "monitored list"). Each window in this list contains a display and identifier that together uniquely identify a window on an X server. For each window, XiVE's update mechanism grabs the contents of the window from the X server and stores its pixel representation in memory. This pixel representation is converted into an OpenGL texture and rendered onto a rectangle in virtual space (see Figure 4.3) [261]. Each window has a transformation matrix associated with it so that a window can be placed at any location, orientation and scale in virtual space, integrated with the other content generated by the VE. Once the windows to monitor are identified, XiVE repeatedly grabs their contents and renders them as textures in the VE. At the same time, XiVE monitors the VE's interaction devices for intersection with the displayed textures and manipulator events (such as button presses). When this happens, XiVE generates synthetic motion, button and keyboard events (using the XTEST extension in the X server) to mimic the behavior of a conventional mouse or keyboard input device.

Identifying windows and displays

Windows and displays are identified by the hostname of the system running the X server and a numeral to identify the display, screen, window or a shared memory

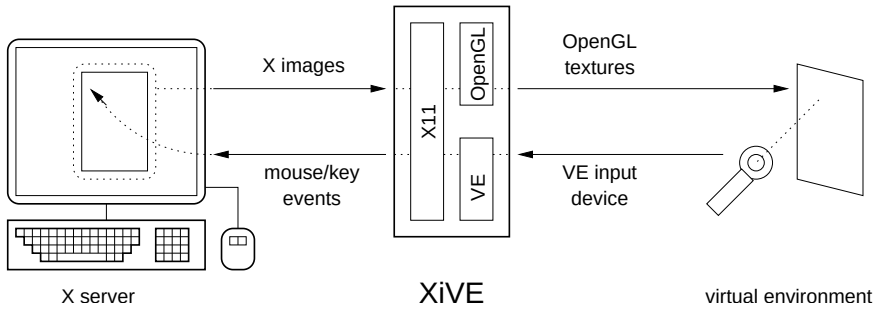


Figure 4.3: XiVE *grabs window images from X servers and renders them in the virtual environment as OpenGL textures. Interaction with the applications is made possible through synthetic keyboard and mouse events using the* XTEST *extension of the X server.*

identifier, using the following notation:

`[hostname]:display[.screen][#window|%memid]`.

The members between square brackets are optional; if `hostname` is empty, the local host is assumed; if `screen` is empty, the default screen of the display is used; if `window` is empty, all windows on the display are used; if `memid` is given, a shared memory rendering X server is used. Using this identifier, windows and displays on an X server can be rendered in a VE using any combination of the following configurations (see also Figure 4.4):

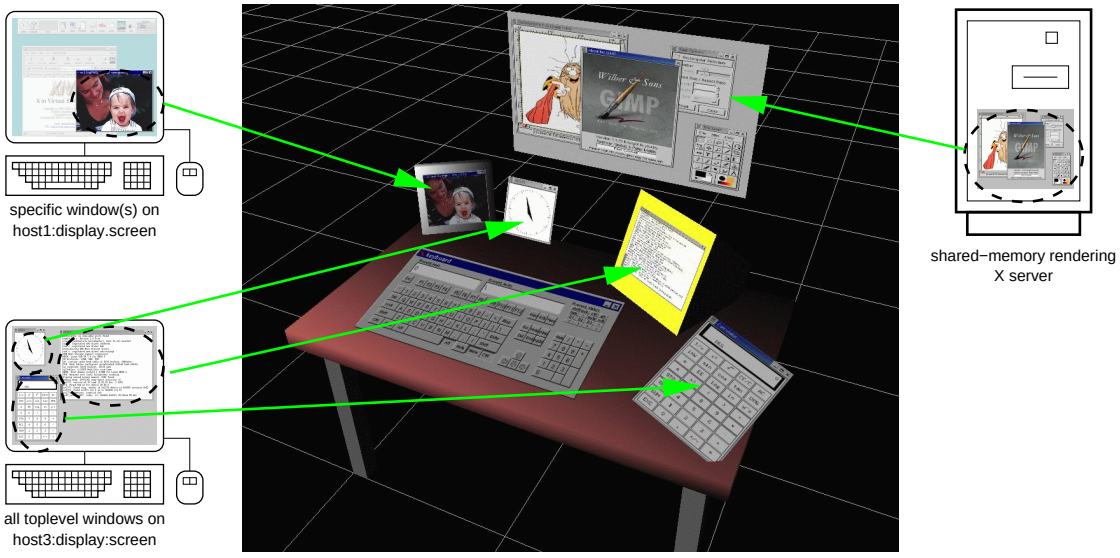


Figure 4.4: XiVE *can monitor windows from specific windows on an X server (top left), all top-level windows on an X server (bottom left), or shared-memory rendering X servers (top right). See also colour reproduction on the back cover.*

1. **Specific windows** - For example: `dexter:2#0x23` which identifies window `0x23` on the default screen of display `:2` on host `dexter`. The window is represented by a textured rectangle in the VE. Note that the "root window" on an X server is a special window, represented by a unique window identifier, that contains all other windows on the screen.

2. **Top-level windows** - For example: `dexter:0.2` which identifies all top-level windows on screen 2 of display `:0` on host `dexter`. Each window is represented by a textured rectangle in the VE. The initial placement of each new window mimicks that on the X server so that overlapping windows on the X server overlap in the same way in the VE. This ensures that pull-down menus and pop-up windows appear at the same relative location. This initial placement can be overridden using the transformation matrix so that, for example, the window can be moved elsewhere through user interaction.

3. **Shared memory rendering X server** - For example: `:9%12345` which identifies a shared memory rendering X server running on the local host with display number `:9` and shared memory id `12345`. The root window of a shared memory rendering X server is represented as a single textured rectangle. The window image can directly be accessed by a XiVE application, since it is located on the same machine, so that it does not have to be copied using a network transfer which improves update rate. The display identifier is required in this case only to enable interaction with the client applications rendering on the X server. An example of an X server that renders into memory instead of on a graphics display is `Xvfb`, which comes with most X distributions [140].

4.3.3 Performance issues

Two factors characterize the performance of a XiVE application. The first is *update rate*; the frequency at which new content of an X window is represented in the VE. Low update rates may result in important information being missed while at the same time it may make interaction with the application difficult, especially when visual feedback is of importance during the interaction with the X application. The second is *frame rate*; the frequency at which a representation of an X window can be rendered in the VE. Since most immersive environments render the environment from a user-centered point-of-view (using the position and orientation of the user), the frame rate should be at least 10 frames per second for a responsive environment. Low frame rate results in decreased response from the environment which in fully immersive systems can cause motion sickness or at the very least negatively influences interaction with the VE.
The main factors that limit performance are (1) the limited performance of the network connection between the host running the XiVE application and the host running the X server, (2) the delay caused by the conversion of a grabbed image into a representation that can be used as input format for OpenGL textures, (3) the delay caused by the definition of the texture, and (4) the time that is required to render a textured

rectangle in the VE. XiVE uses a number of techniques to reduce these overheads during the four steps that are required to update a representation of a window in the VE:

1. **Obtaining a window image from the X server** - In a situation where the X server is not running locally, the image must be obtained using a network transfer. High network latency reduces response time, low bandwidth reduces the frequency of updates that can be rendered by XiVE. These network limitations can be reduced by executing an X server on the same machine as where XiVE is used so that communication takes place internally, or by using an X server that renders images into shared memory which can be accessed without any network communication.

2. **Conversion of an image to a format suitable for texture rendering** - Images retrieved from an X server are stored in color indexed format so that every pixel is represented by a value that represents a color through a lookup table. OpenGL version 1.1 and later supports this kind of image format in different flavors and XiVE attempts to detect which are supported by the implementation of OpenGL. If no support for color indexed textures is detected, XiVE falls down to an algorithm that converts the color indexed images to RGB format which is supported by all OpenGL implementations. The latter format takes time to generate and requires considerable more memory to store and so XiVE benefits from OpenGL libraries that have support for color indexed textures.

3. **Texture definition** - Defining textures in OpenGL often involves downloading the texture into dedicated texture memory on a graphics interface. To avoid unnecessary overhead, XiVE detects whether a window's contents has changed before redefining its texture. Although this requires that a new image is obtained from the X server, this update scheme executes in a separate thread of execution in XiVE. This thread detaches from the rendering processes and constantly monitors the windows maintained by XiVE and redefines a texture only when the contents of a window has changed.

4. **Texture rendering** - Texture rendering is one of the most expensive operations in computer graphics. XiVE benefits greatly from graphics hardware that is capable of performing texture mapping in hardware.

Performance results

Tables 4.1, 4.2 and 4.3 illustrate the overhead in these four steps for a profiled XiVE application running on a Sun UltraSPARC-IIi at 300 MHz, running Solaris 8 and equipped with an OpenGL 1.2 accelerated Elite3D graphics interface. The XiVE application grabs the root window from an X server running an X server at 800 by 600 with 16 bit pixels (937.5 kB total image size) and renders this window using either RGB or color indexed (CI) textures.

	RGB (time)	(%)	CI (time)	(%)
image transfer	1040 ms	49.7%	1040 ms	58.7%
conversion	750 ms	35.9%	430 ms	24.3%
texture definition	300 ms	14.3%	300 ms	16.9%
rendering	1 ms	0.0%	1 ms	0.0%
total	2091 ms		1771 ms	

Table 4.1: Overhead of the four steps to update a window representation using a 10 Mbit/s ethernet connection between XiVE *and the X server.*

In Table 4.1 the two systems are connected via a 10 Mbit/s ethernet. The table shows that most overhead is caused by the transfer of the window image from the X server. Since the update rate is directly related to how fast images can be obtained from the X server, this configuration is useful only in cases where update rate is not of high importance. Note that the time to render the window is only 1 ms which leaves ample time to render other geometry in the VE and still maintain a sufficiently high refresh rate. The table also shows that the time required to convert the transfered image into a suitable format for texture rendering is significantly lower in the case of color indexed textures.

	RGB (time)	(%)	CI (time)	(%)
image transfer	36 ms	3.4%	36 ms	4.7%
conversion	730 ms	68.4%	430 ms	56.1%
texture definition	300 ms	28.1%	300 ms	39.1%
rendering	1 ms	0.0%	1 ms	0.0%
total	1067 ms		767 ms	

Table 4.2: Overhead to update a window representation using internal network communication.

In situations where high update rates are critical, the transfer overhead can be significantly reduced by running the X server on the local host, as shown in table 4.2. As the X server in this situation is running on the same host as the XiVE application, all image transfers occur using internal communication. The table clearly shows that the transfer overhead is significantly reduced. In fact, this overhead can be completely canceled through an X server that renders into shared memory; as XiVE can access this memory directly, no images need to be transferred at all, as can be seen from Table 4.3.

4.3.4 Conclusions

We have described an architecture that allows existing applications with 2D graphical user interfaces to be used in 3D immersive virtual environments. This architecture bridges a gap between the 2D user interface libraries that exist for desktop application development environments and VE development environments that in general

	RGB (time)	(%)	CI (time)	(%)
image transfer	0 ms	0.0%	0 ms	0.0%
conversion	730 ms	70.8%	430 ms	58.8%
texture definition	300 ms	29.1%	300 ms	41.0%
rendering	1 ms	0.0%	1 ms	0.0%
total	1031 ms		731 ms	

Table 4.3: Overhead to update a window representation using a shared-memory rendering X server.

lack these facilities. Through XiVE, VE application users are presented with user interfaces and applications they know well from conventional desktop applications which greatly facilitates interaction in the VE and may result in increased productivity. The design of XiVE allows existing VE applications to be easily extended with this feature at little loss of performance. For example, Figure 4.5 shows XiVE used in the VRE application described earlier in section 2.3 (page 29) to set a threshold for the isosurface modeling visualization pipeline via an interface designed in Tk [175].

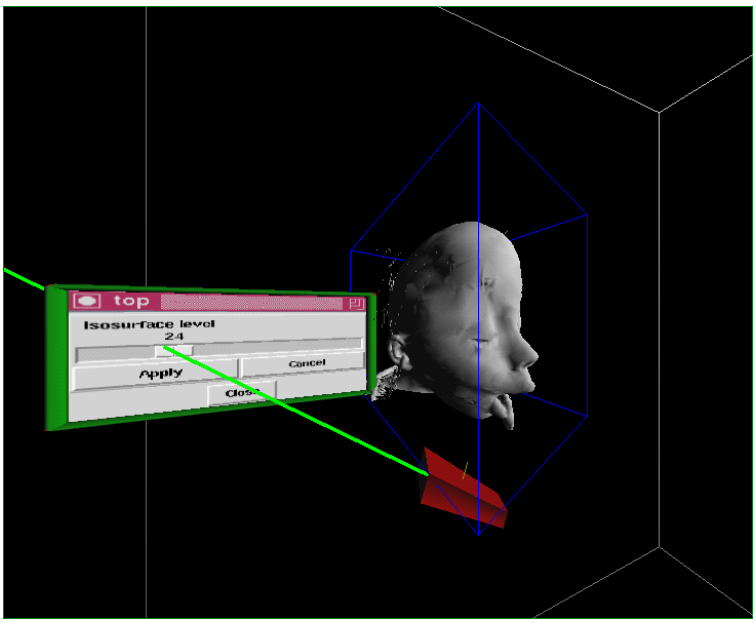


Figure 4.5: XiVE used in the VRE environment (see section 2.3) to set an isosurface threshold via a user interface defined in Tk [175].

4.4 Context sensitive interaction

The applicability of an interaction method is largely determined by its degrees of freedom, in providing sufficient expressiveness so that multiple functions can be triggered by similar but different gestures, and by its robustness to discern one gesture from

another. With the increase in functionality that can be triggered in an application using an interaction methodology, the chances increase that a user triggers a function that was not intended. A relatively unexplored paradigm for interaction in a VE that can be used to reduce this, is to restrict whether gestures are recognized based on the context of the application. Although it may seem contradictory to constrain a user in a restricted set of gestures, the aim of the technique described here is to provide the user with more efficient, goal-directed interaction methods by eliminating those that are out of context [23]. In the following we will describe the concept of "application context" in a VE and how context can be used to improve the efficiency of an interaction method.

4.4.1 Determining application context

During the use of a VE, the environment takes on a new state as the user interacts with the environment. Each change as a result of user interaction or autonomously operating entities in the environment, alters the state of the environment, potentially bringing the user into a different context. Accurate determination of context is not always a trivial task. In general, properties that have the most significant impact on context and that can be easily obtained from the VE are:

- The state of entities in the environment - The existence or properties of entities in the environment imply that there is a chance that the user may want to interact with the entities or its properties. Therefore, the interaction methods supporting the interaction on this entity should be enabled. Likewise, if entities or properties are removed, its associated interaction methods may be disabled.
- The user's focus of attention - When a user is located in the vicinity of, or has his focus on a certain area in the environment (either by looking at, being in the vicinity of, or pointing at entities in the VE), there is a chance that this user will want to perform interaction on entities in this area, and not on those which are, for example, out of view.

Determining these properties is possible if the VE provides access to the data structures that determine them.
Enabling every possible interaction method in the environment may result in incorrect interpretations of the user's intentions and may therefore result in an unintended response from the environment. By disabling interaction methods that are not applicable to the context of the environment at some particular moment, the chances that gestures are mistakingly recognized for something which was not intended can be decreased.

The context engine

We have built a "context engine" that monitors the state of the environment asynchronously [267]. This context engine provides information to "agents" that dynamically adapt the interaction methods enabled by the environment at any time. Based

on that, interaction methods are enabled or disabled automatically as the context of the VE changes. Still subject to ongoing research at the University of Amsterdam, the agents have the ability to perceive state changes through monitors, take actions that affect conditions in the environment and perform reasoning to interpret perceptions, solve problems and determine actions. As a test case, we have applied the context engine to improve the reliability of interaction via speech recognition.

4.4.2 Context sensitive speech recognition

Speech recognition involves the transformation of an acoustic speech signal into written text. Development of speech recognition has been ongoing for over 25 years and only recently speech applications are coming into widespread use. Speech recognition can be very useful in applications where hands and eyes are constantly busy, which is often the case in immersive environments, or for people with disabilities that inhibit their motor skills. Psychological studies have shown that users of virtual environments prefer to use a combination of speech and hand gestures so that they can concentrate on the objects in the virtual environments and the task at hand [96, 97]. Also, speech input does not require special training of the user or unfamiliar input devices. Moreover, since speech recognition systems operate on a different input modality (namely sound), they are not hampered by limitations of graphics resources, thereby making it a suitable alternative input modality for situations where visual fidelity (such as frame rate) is critical.

Common problems with speech recognition

Acceptance of speech technology is growing although there are still some key issues holding it back, i.e. recognition accuracy and ambiguity [42, 177, 208, 228]. Common situations where accuracy is important is where the speech recognition system must be able to discern words that have approximately the same pronunciation (like “dye” and “die”). Ambiguity occurs when the same verbal command is used to trigger different functionalities. For example, consider an application that supports a “volume” command. In one particular context this command could refer to the output gain of an audio device while in another it could refer to a quantitative property of a three-dimensional object. To resolve this ambiguity, the user would have to add extra information (e.g. “audio volume” or “object volume”) in order to unambiguously identify the target for this command. Although the recognition will perform better with this extra information added, the user is expected to have a thorough understanding of the speech recognition system’s grammar. This will not make the system easier to use. These problems can be reduced by incorporating application context into the speech recognition systems.

Related work

In 1980, Bolt has shown the added benefit of interaction through a combination of speech and context information derived from hand gestures in his “Put That There”

demonstration system [21]. The application and benefits of speech input in visualization and virtual environments is described in [215, 239]. Oviatt et al. integrate complementary modalities in a manner that supports mutual disambiguation of errors to improve performance [177, 263]. Suhm et al. describe a multimodal error correction that uses context provided by additional modalities (like key, mouse and pen input gestures) to correct errors [228].

Applying context to speech recognition

Context sensitive speech recognition (CSSR) systems can have large vocabularies (i.e. thousands of words), but only a subset of that vocabulary is activated at a particular time. The complete vocabulary is subdivided into subsets that contain commands that are applicable to the context for which they are intended. While the application is running, the context engine determines the context of the application and enables only those subsets of the vocabulary that are applicable to the application context. As only a part of the complete vocabulary is enabled, the chance that a verbal command is incorrectly recognized is reduced, thereby increasing the overall accuracy of the speech recognition system. Note the resemblance with conventional desktop applications where the user is also frequently inhibited from invoking some functionality that are "inappropriate" by disabling the options.

Performance results

A speech recognition library capable of context sensitivity (CAVETalk) has been developed. The core of CAVETalk uses IBM's ViaVoice as third-party recognition software [47, 104, 105]. ViaVoice is a speaker independent speech recognition system that can handle isolated or continuous recognition. User specific training is not necessary, but can be done to improve recognition accuracy. Vocabularies in ViaVoice are defined in Backus-Naur Form (BNF). Using this form, it is possible to define extremely large and flexible vocabularies in a very concise notation, as shown in an example in Figure 4.6.

An experiment was conducted in which the user was presented with a verbal command set that consisted of 57 words in total, divided into five subsets. The user was asked to issue the commands in random order, as provided by an external randomization program. During a single experiment, the user was asked to issue each command twice; once when all words in the vocabulary would be enabled, the second time only the subset of the vocabulary that contained the command would be enabled. The user did not know whether the whole vocabulary or only a subset would be enabled.

In total, 13 experiments were performed. In the non-context situation, the mean number of correctly recognized commands was 45.2 (79.4%) with a standard deviation of 4.4. In the context situation, the mean number of correctly recognized commands was 49.5 (86.8%) with a standard deviation of 4.1, an average improvement of 7.4%. Although this improvement may seem marginal, the vocabularies in this experiment were intentionally chosen to contain words that had little similarity in pronunciation. For vocabularies that contain words with a similar pronunciation the results

```
<<start>>
  = "create" "a"? <target> <destination>?
  | "delete" <denom>? <target> <source>?
  | "move" <denom>? <target> <source>? <destination>
  | <quit> .

<target> = "sphere" | "cube" .

<denom> = "the" | "that" | "this" .

<source> = "at the" <location> "of" <denom>? <target> .

<destination>
  = "here"
  | "to the"? <direction>
  | "to the" <location> "of" <denom>? <target> .

<direction> = "up" | "down" | "left" | "right" | "front" | "back" .

<location> = "top" | "bottom" | "left" | "right" | "front" | "back" .

<quit> = "quit" <program>? | "exit" <program>?  .

<program> = "program" | "application" .
```

"create cube", "create a sphere", "create a cube to the left of the sphere", "create a sphere here", "move cube right", "move that cube to the back of this sphere", "move the cube at the left of that sphere to the right of this cube", "move the cube up", "move that sphere here", "delete the cube", "delete the sphere at the left of that cube", "exit program", "quit".

Figure 4.6: Example of a vocabulary in BNF notation used by the ViaVoice speech recognition system. With this vocabulary, 24558 different sentences can be recognized, some of which are shown at the bottom.

are much higher, provided these words are in different subsets and are not activated at the same time.

User reactions

In the environments we developed, users are able to navigate through the VE with great ease by pointing at objects in the environment and saying "come here" or "go there". The use of context also dismissed the need for "push-to-talk" solutions where the voice recognition system's attention is explicitly told when to listen to verbal commands [142], as the system only listens when there is clear defined context. This allows users to freely discuss the things that are going on in the environment with

their peers, without having to worry that the environment responds unintentionally.

4.5 GEOPROVE: Geometric Probes for Virtual Environments

In many scientific computing problems, the level of complexity in the generated data is too vast to analyse numerically. For these situations, interactive scientific visualization is an essential method to present and explore the data in a way that allows a researcher to comprehend the information it contains. Immersive virtual environments such as the CAVE [52] further enhance a researcher's perception and are therefore often used to obtain better insight in multi-dimensional datasets for which desktop visualization environments are too restrictive.
In most scientific visualization environments, a visualization pipeline transforms numerical data into geometric constructs that are rendered into visual presentations. These presentations allow researchers to qualitatively analyse their data. Many visualization environments stop at this point and provide little means to obtain quantitative information on what is being presented. Through the use of stereoscopic images it is possible to estimate quantitative properties, such as the (relative) size and distance of virtual objects [200]. This may be acceptable to some applications, for others however, an instrument for obtaining quantitative information from the visualization is a valuable asset. Examples of this are applications where simulations are verified to the real-life phenomena that are being modeled, applications for diagnostic purposes based on medical data obtained from medical scanners (i.e. CT, MRI, etc.), or computer aided design tasks.
In this section we describe GEOPROVE, a geometric probing software architecture for interactive data exploration environments, virtual environments in particular. This architecture allows researchers to probe visual presentations in order to obtain quantitative information. We will show the application of this probing system with the test-case described earlier in section 2.4 (page 42).

4.5.1 Related work

While most scientific visualization environments provide some probing functionality, most of these act as subset selectors that extract selected regions from larger data sets for localized visualization, complementing global visualization methods [172, 242]. The Visualization Toolkit (Vtk) for example provides probe filters for the computation of point attributes in local areas. Point attributes are computed at input points specified by a probe consisting of a geometric structure by interpolating into the source data. Vtk also contains methods that calculate properties such as the volume, surface area and normalized shape index of closed triangle surfaces [209].
The work by van Leeuw et al., takes this method one step further by using visual probes that consist of a set of geometric primitives. Multiple characteristics of a small

area in a flow field transform the geometric primitives in the probe to visualize velocity and local change of velocity [56]. Although these visual probes provide excellent means for exploring local properties of a dataset, they are often used for localized visualization only, and not for obtaining quantitative measurements.
The work by Brady et al., shows a CAVE application for the visualization of biomedical images obtained from medical scanners that allows features to be manually traced and labeled [25]. This software has been used for obtaining the lengths of biological structures and for segmenting medical images.
Germans et al. describe an approach to obtain measurements from the visual domain which is similar to what we describe here [194].

4.5.2 Geometric probing

In our system, geometric probes consist of one or more markers. These markers are used to sample properties of the presentations in the virtual environment. A property can either be the coordinates of a position in the environment, or a value obtained from data at this position. The property obtained from the markers are used in an evaluation function producing the result of a measurement. The evaluation function defines a relation between markers in a probe, which we illustrate here by connections between markers.
Determined by the spatial configuration of the markers, probes have dimensions and a certain degree of freedom by which they can be positioned over an area of interest. For example, a probe consisting of exactly one marker has 3 degrees of freedom in a 3D environment (translation in 3 directions) and can thus either be used to obtain the position (x,y,z) of a feature in 3D space or a mapped quantity $f(x,y,z)$ at this position, where f provides a mapping of a position to a quantity (i.e. a scalar, vector, tensor). An evaluation function takes this sample and produces the result of the measurement.

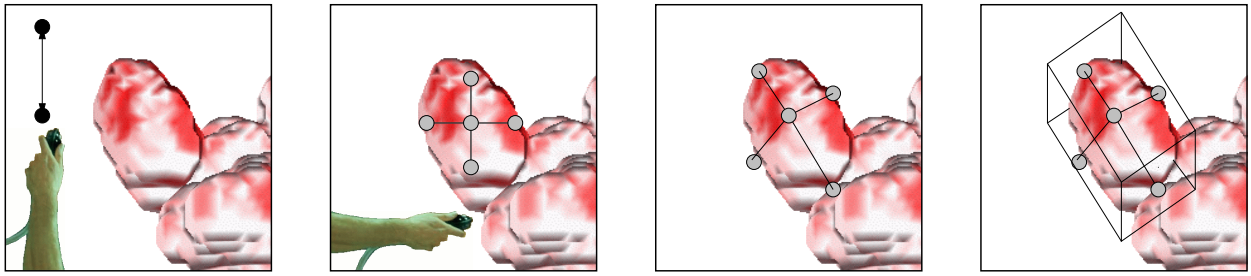


Figure 4.7: Probing procedure, as an example applied here to determine the bounding volume of part of a structure. From left to right: calibration, interactive placement of the probe, registration of the probe to the data, calculation of bounding box.

A measurement procedure with probes in a virtual environment requires the following course of actions (see also Figure 4.7):

1. Calibration - As in any measurement, the properties that are calculated based on sampled quantities need to be referenced to a well defined unit. This unit

of reference is especially required when the measurements from two different objects are to be compared. For the same type of measurements, calibration will only have to be performed once.

2. Placement of the probe - Interactive placement of a probe in a virtual environment makes use of devices whose position and orientation are tracked in three-dimensional space. Through these devices, a user is able to place a probe roughly over the region of interest.

3. Registration of the probe - The interactive placement of a probe is not accurate in most cases because of inaccuracies in the tracking hardware or inexperience of the user. Registration of the probe involves refining the position, scale and rotation of the probe, either interactively or aided through registration functions.

4. Calculation of the result - Once the probe is in place, the result can be calculated. Depending on the type of probe, the calculation is either performed purely based on the position and orientation of the probe, or the positions of each marker are first mapped to a quantity.

5. Presentation of the result - When the calculation is finished, the results need to be presented to the user in some meaningful way. In addition, the user should be able to log the measurement on file for later analysis and some method of annotation is required so that the user can relate back to the measurement once they are analysed elsewhere.

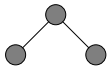
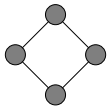
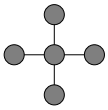
# markers	1	2	3	4	5
probe					
positional property	position	length, distance	angle, curvature	bounding rectangle	saddle point
mapped property	value	derivative	extrema	surface	Gaussian distribution

Table 4.4: Examples of probes with different number of markers and examples of positional and mapped properties that can be measured with these probes.

Table 4.4 shows some examples of probes consisting of a number of markers and examples of properties that can be obtained. Most of the properties in this table can be relatively easily obtained. For the first implementation of this architecture, with the coral growth data exploration system as a test-case, we limit ourselves to measurements that require probes consisting of 2 markers (length, distance) and 3 markers (angles). However, in the design of GEOPROVE we have attempted to keep the architecture generic so that the addition of probes for the acquisition of other properties can be achieved with little effort.

Software architecture

Interactive visualization applications benefit from a design in which the computation and visualization processes are implemented by separate communicating threads of control [31]. The library used in CAVE environments supports primitives for this design [249]. If implemented carefully, this configuration allows interactive virtual environments to be built that have a high frame rate and minimal interaction delay. However, it does have implications for the design of the GEOPROVE architecture. The software architecture that we have developed is shown schematically in Figure 4.8.

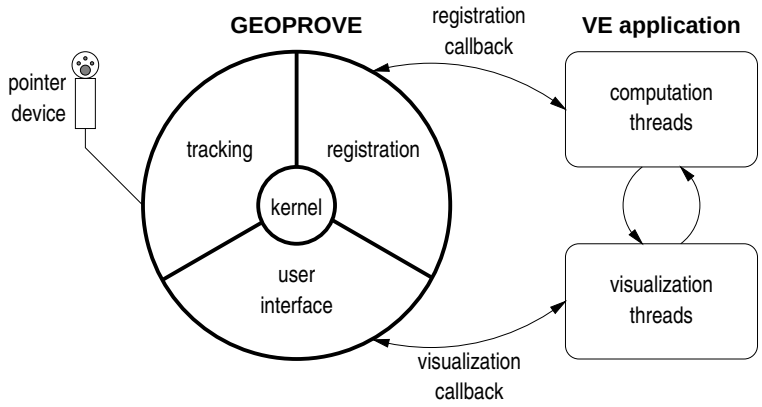


Figure 4.8: The software architecture for GEOPROVE *and its interface to a virtual environment application.*

The heart of GEOPROVE consists of the *kernel* that logs the positions and annotations of the probes, performs calculations and maintains a record of the measurements for storage and retrieval. The *tracking* part provides the kernel with position and orientation information of pointer devices with which probes can be positioned in the virtual world. The *registration* part manages the positioning of probes based on information provided by the application. This allows probes to be "snapped" in real time to computed data so that accurate measurements can be obtained. The *user interface* part provides access to GEOPROVE, including the rendering of probes, the presentation of results and annotations, and methods to drive GEOPROVE.
Measurements can be obtained on two levels; on the first, most basic level, measurements are based purely on the position and orientation of pointer devices that are tracked in three-dimensional space. On the second level, measurements take place based on structures defined by the application. This two-level scheme is described in more detail in the following two sections.

Tracker based probing

On a most basic level, probes are positioned based solely on the location of pointer devices that are being tracked in the visualization environment domain. Note that in this situation measurements are taken without a direct relation to the application other than the user's perception of the environment. This allows simple mea-

surements to be made such as distances and angles without any feedback from the application.
As this method of probing is independent of what is being presented, probes can not be registered on data from the application. Therefore, its accuracy and usefulness is inherently dependent on the quality of the tracking systems, the experience of the user and the application for which it is used. Most tracking systems are sensitive to noise, therefore GEOPROVE supports scaling the coordinates obtained from the tracker system such that a more accurate positioning of a probe can be performed. In this research, the physical devices that are tracked consist of a head tracker and a "wand", a device that is similar to the desktop mouse but which is tracked using a six degrees-of-freedom sensor [205].
The main benefit for this kind of probing lies in its simplicity and its allowance for the fast acquisition of positional measurements such as lengths, angles, spatial derivative approximations and special geometries like the fractal box dimension [78]. Furthermore, as the implementation of this kind of probing can be isolated into GEOPROVE itself, it minimally interferes with existing VE software.

Mapping markers to quantities

Quantities that relate to properties that are defined by the application can only be obtained by interrogating the visualization or computation thread. Since GEOPROVE does not have direct access to the data maintained in the application, the quantification of a marker from its position has to be handled by the application via a callback function.
Some quantities can be best obtained from the abstractions that are made from application data when these are visualized. An example of this is the calculation of a surface area which can be approximated using isosurfaces that are extracted from grid based data for visualization through e.g. a surface extraction algorithm. For other measurements, the data contained in the computation may be of higher quality.

Registration

For some types of measurements it may be necessary to perform calculations on specific features of the underlying data. In these cases the probe needs to be aligned to these features before calculations can be performed. Depending on the probe's shape and its degrees of freedom, the registration of a probe to the underlying data sets takes the same form as the techniques that are used in *geometric hashing* [260]. In short, this method first takes two markers of the probe as a handle to match "points of interest" in the geometric dataset. Using geometric transformations (the most common being translation, rotation and scaling) a basis is constructed which determines the exact position of all markers in the probe. Through a voting mechanism a histogram is then constructed of candidate bases from which the best candidate for registration is chosen. Using this method, positions acquired from tracker sensors can be registered to data structures that have been used for visualization or computation.

Although this technique is in general computationally expensive, we may exploit "focus locality" by disregarding the geometry which is far from the user's focus.

Presentation, logging and annotation

GEOPROVE does not render its own user-interface. Instead, it relies on the VE application to do this. This allows GEOPROVE to be seamlessly integrated into existing applications. Most of our applications contain standard user interface components that can be used by multiple processes at the same time (see also Figure 4.10). Existing software can be instrumented with probing facilities with minimal effort. In the current version, only four function calls need to be added to the source code of an existing application to obtain the most rudimentary features. These functions consist of: (1) the initialization of GEOPROVE, (2) a display handler that renders feedback to the user for both interaction and presentation of results, (3) a user interaction handler, and (4) a registration function that allows markers to be snapped to visualized geometry.
Probe locations and measurement results are stored in a log file. During runtime, the user can browse through this log via the user interface and view entries or delete entries. The user also has the option to add annotations to a measurement via a "snapshot"; a virtual photograph made from the perspective of the user. Both low resolution (for runtime inspection) and high resolution (for off-line inspection) pictures are supported. We are currently adding the option to record a speech-sample to annotate measurements with extra information.
When the application is terminated, the log is written to a file, containing the measurements and references to possible snapshots and speech annotations. The log can then be analysed on a workstation, saving valuable CAVE time.

4.5.3 Position accuracy of the SARA CAVE tracking sensors

We have performed experiments to measure the accuracy of the tracker installation in the SARA CAVE. We are not so much concerned with the difference between the reported tracker position and the physical position of the tracker, since this can (at least partially) be corrected with visual feedback (e.g. by drawing a cursor at the reported tracker position). What is important for a probing system is the ability for users to indicate a certain position in 3D space. The accuracy with which this can be done depends on several factors: the resolution of the tracker system, the quality of the visualization system, and the skill of the user.
Our measurements are based on a task where a sphere was drawn at one of 196 target positions, and another sphere (drawn at the reported position of the wand) had to be 'superimposed'. Figure 4.9 shows a visual representation of our results. These measurements were obtained by sampling the 196 target positions over 6 independent experiments by two experienced CAVE users. Each target position was sampled between 10 to 15 times. These measurements clearly show that large deviations from target positions take place most often in the corners and at the entrance of the CAVE,

making it almost impossible to accurately position a marker at these locations. We have observed "jumping" behaviour in the reported position of the wand where slight changes in hand position result in very different reported positions from the tracker. The worst deviation we have measured was as high as 47 cm. This error can be attributed to a large concrete pillar at the entrance of the CAVE which is far away from the magnetic source and very likely contains metallic reinforcements that distort the magnetic field.

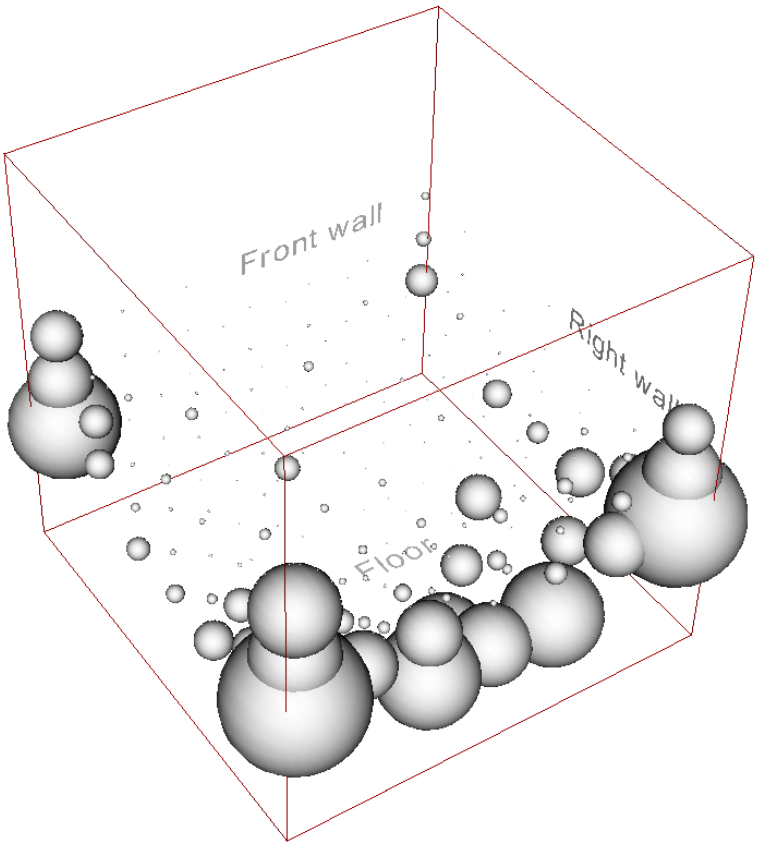


Figure 4.9: Mean deviation of reported tracker positions against target positions in the SARA CAVE.

To compensate for these large errors, we have opted for a refinement system in which the user is able to refine initial probe positions through down-scaling the tracker output. This way, large displacements of the hand result in smaller displacements of the probe, allowing probe positions to be refined irrespective of tracker hardware restrictions.

The best results are obtained in the center of the CAVE. Here target positions can be indicated with very small deviations (< 1 cm). Ways to increase the accuracy of the tracker systems used in projection-based VR systems such as the CAVE are reported in [54].

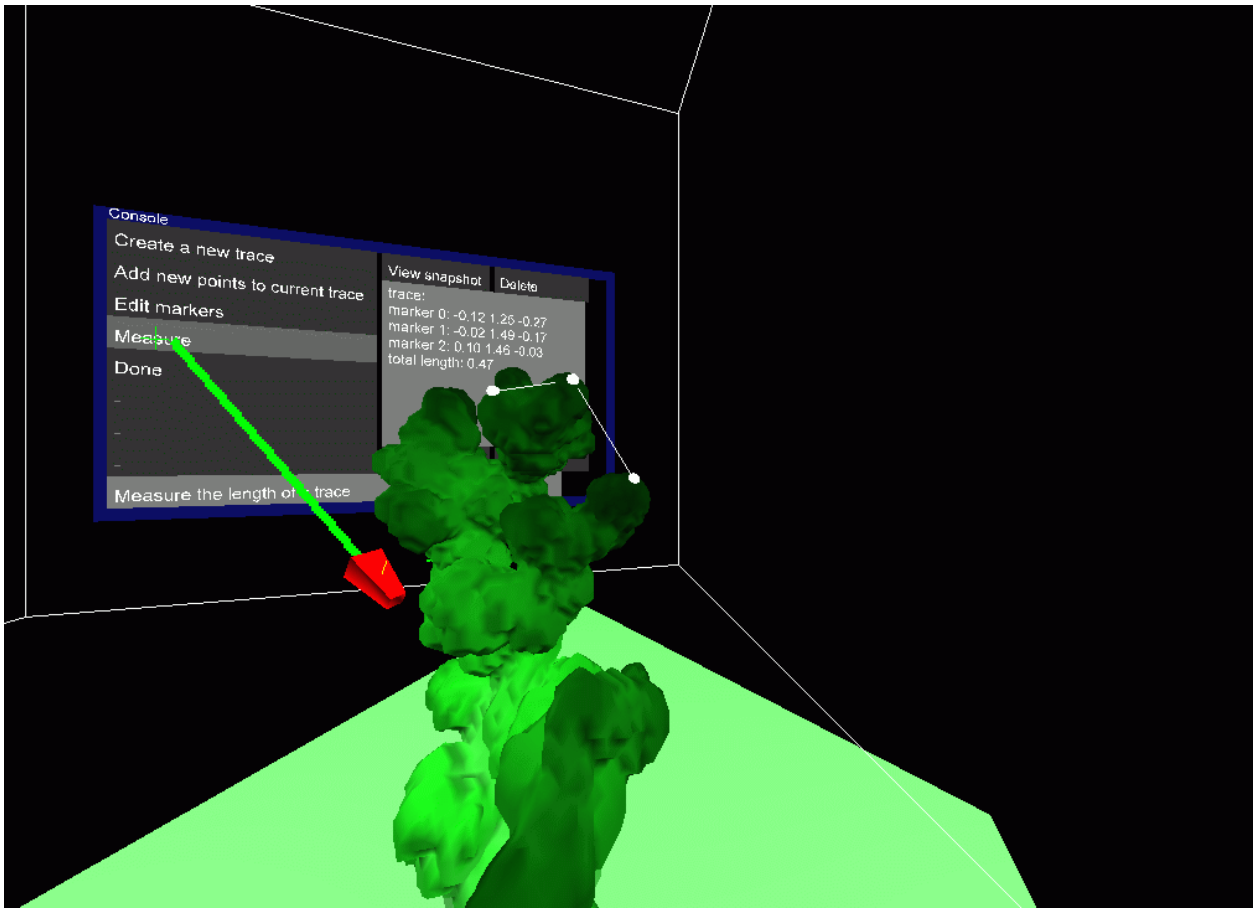


Figure 4.10: A CAVE simulator snapshot of an application instrumented with GEOPROVE. *Three markers (shown in white) of a trace are registered (or "snapped") to the visual geometry. The window in the back shows the user interface to* GEOPROVE *and presents the length of the trace.*

4.5.4 Results

We instrumented the interactive coral growth exploration environment described in section 2.4 (page 42) with GEOPROVE to obtain measurements of the shortest distances between neighbouring branch ends ("branch spacing") (see also Figure 4.10) [13]. The importance of this spatial observable is described in [211].
Figure 4.11 shows histograms of 106 measurements obtained from a CT scan of a real coral structure (*Pocillopora damicornis* in a sheltered environment) and 155 measurements obtained from 8 simulated coral structures under similar conditions.
It can be observed from the measurements shown in Figure 4.11 that there is a remarkable difference between the measurements of the branch spacing done in the CT scan of *Pocillopora damicornis* and the simulated growth forms. Three observations can be made from these measurements:
(1) The measurements from the simulated structures seem to be bimodally distributed while the CT scan measurements are unimodal. This may be the result of mea-

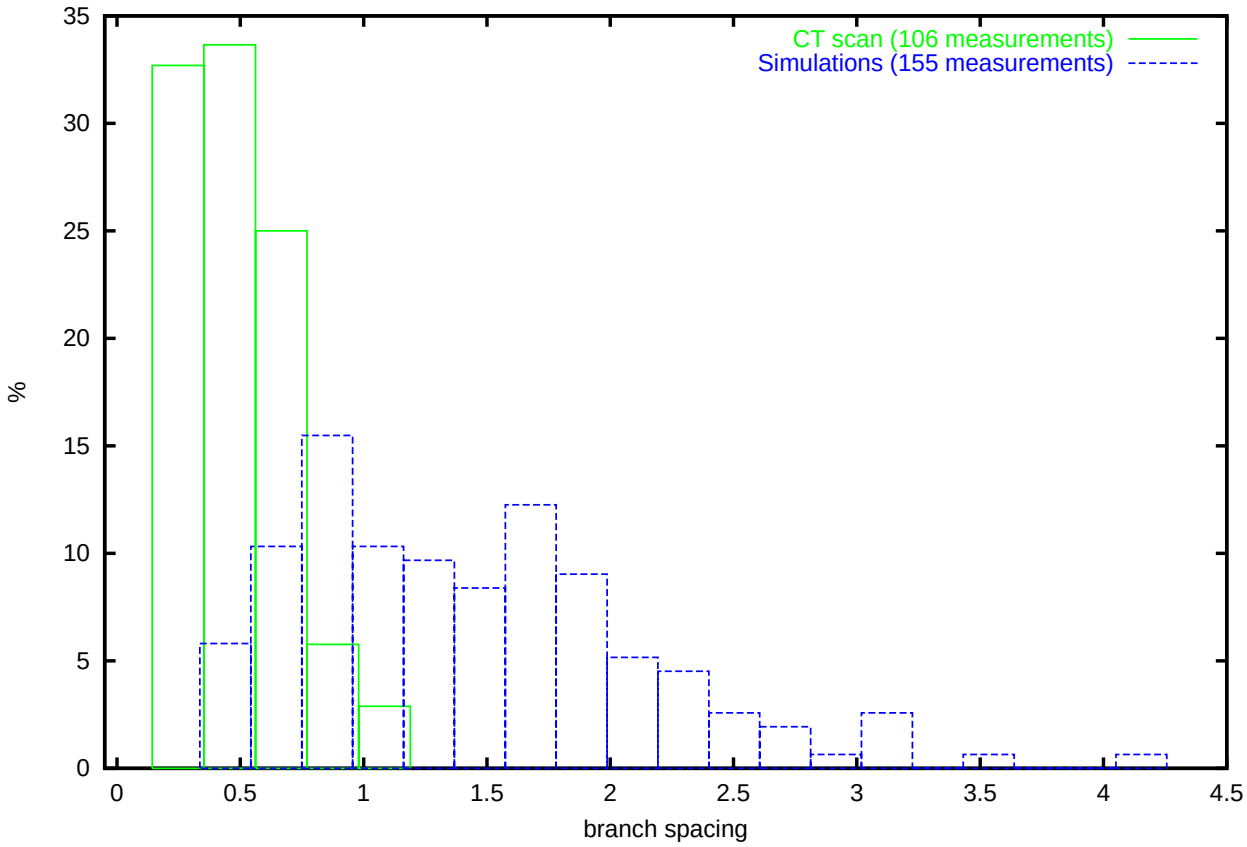


Figure 4.11: Comparison of branch spacing in a CT scan of Pocillopora damicornis *in a sheltered environment with 8 simulated structures under similar conditions.*

surement artifacts or it may be caused by the simulation mechanics. To fully understand the reasons for this we would need to obtain a larger number of more detailed measurements, which we have not yet been able to acquire.
(2) The mean branch spacing of the two distributions differs significantly (≈ 0.5 versus ≈ 1.5). This may be due to the scaling of the measurements: in order to obtain a scale that makes the simulated structures comparable to the real objects, the measurements from the simulated structures have been scaled using a factor obtained from the ratio in dimensions of the simulated structures and the real coral.
(3) In real *Pocillopora damicornis* the variability is relatively low compared to the simulated forms. These measurements seem to indicate that there is a mechanism which regulates growth of branches in the immediate vicinity of other branches, this mechanism is not present in the current simulation models. In the study by Rinkevich and Loya [196] it is proposed for the branching stony coral *Stylopora pistillata* that there is a chemical signal mechanism which regulates growth of branches, resulting in a relatively low variability and even a remarkable uniformity of the branch spacing. Their experiments indicate the possible appearance of a chemical signal which is being secreted into the water column and works as repellent, growth suppress-

ing, agent. A similar mechanism might be present in *Pocillopora damicornis*. The morphological measurements indicate that in future versions of simulation models of stony corals a regulation mechanism is required in order to obtain a better approximation of the actual growth process.

4.5.5 Conclusions

We have presented a software architecture that allows us to instrument interactive virtual exploration systems with probes to obtain quantitative information based on visual presentations. We have used this system to instrument an existing application with little effort. We have limited our system to simple measurements with probes that consist of one, two or three markers. It is relatively easy to extend this system with probes that consist of more complex markers, enabling more advanced measurements to be performed.

4.6 Summary and conclusions

This chapter presented several interaction techniques to enrich the immersive experience of a virtual environment. As in the real world, interaction with a virtual environment greatly increases a user's awareness. Our primary goal was to develop techniques for the purpose of scientific exploration, but most can be applied to other types of virtual environments as well.
By adopting the Visualization Toolkit (Vtk) as our basis for the construction of virtual environments using SCAVI, an application developer has immediate access to a wide range of computer graphics, image processing and scientific visualization functions. Our direct object manipulation extensions to Vtk allow flexible scientific exploration environments to be built within a short time span. The resulting interaction capabilities come close to those offered by commercially available VE toolkits such as the WorldToolkit and Performer, but it also lacks important features, such as a scene graph system. The basic interaction methods are relatively simple and are all based on the wand's buttons and joystick, but have proven to be effective for the implementation of expressive interactive environments.
XiVE provides a unique capability for VE application designers by allowing existing 2D GUI toolkits and applications to be integrated into virtual environments. Its general design allows a developer to design graphical user interfaces using the same programming techniques as are used for desktop applications and include them in virtual environments with great ease. XiVE has recently been released to a selected number of VE application developers for evaluation purposes. Based on their experiences and comments, improvements will be incorporated after which the code will be released to the public.
The concept of context in a virtual environment has, as of yet, only been applied to improve the accuracy of a speech recognition system. Results have shown that the performance of speech recognition does improve, though not by much. The added value

of speech recognition in virtual environments, however, is unquestionable. Through the use of a different modality than graphical representations (i.e. sound), interaction with a virtual environment remains possible even when low frame rates make interaction through graphical constructs difficult. However, experiences with the current version of our speech recognition system show that variations in pronounciation between different speakers remain a problem.
GEOPROVE provides interaction methods for obtaining quantitative measurements from visual representations in a virtual environment, which is essential for scientific exploration. Its design allows VE applications to be extended with this functionality with little effort. The current implementation of GEOPROVE provides its own graphical user interface to facilitate integration in existing VE applications, but it has been designed in such as way that this can be replaced by other interfaces.

Chapter 5

Interactive dynamic exploration environments*

"Computing machines can do readily, well, and rapidly many things that are difficult or impossible for man, and men can do readily and well, though not rapidly, many things that are difficult or impossible for computers. That suggests that a symbiotic cooperation, if successful in integrating the positive characteristics of men and computers, would be of great value. The differences in speed and in language, of course, pose difficulties that must be overcome."

J.C.R. Licklider, Man-Computer Symbiosis, IRE Transactions on Human Factors in Electronics (HFE-1)*, pages 4–11, 1960.*

5.1 Introduction

The interactive dynamic exploration environment (IDEE) model illustrated in Figure 1.2 (page 5) is equivalent to a common design pattern known as the *Model-View-Controller* (MVC) architecture (see Figure 5.1) [85]. The MVC architecture has its roots in Smalltalk-80 where it was originally applied to map the traditional input, processing, and output tasks to the graphical user interaction model [136]. This three-way abstraction separates (1) the model of the underlying application, (2) the representation of this model to the user and (3) the ways in which the user interacts with the application. Compared to our IDEE model, the "Model" in the MVC architecture corresponds to our Process (or Simulation) component, "View" corresponds to

*Parts of this chapter have been published in *R.G. Belleman and P.M.A. Sloot. "The Design of Dynamic Exploration Environments for Computational Steering Simulations",* Proceedings of the SGI Users' Conference, pages 57–74, 2000 and *R.G. Belleman and R. Shulakov. "High performance distributed simulation for interactive simulated vascular reconstruction", number 2331 in* Lecture Notes in Computer Science, pages 265–274, 2002.

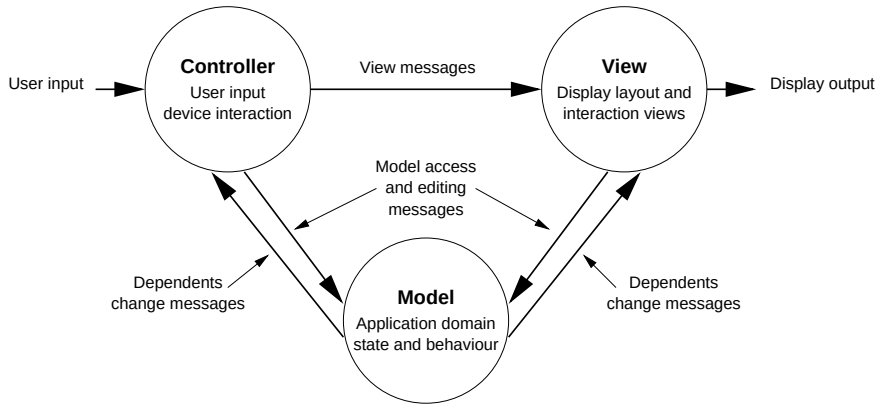


Figure 5.1: State and message handling in the Model-View-Controller *(MVC) design pattern [136].*

Presentation and "Controller" to Interaction.
The rationale behind isolating functional components from each other is two-fold. First; this design helps the application engineer to understand and modify each component without having to know everything about the other components. Here, the *Model* is the central component of the application, the one that does the interesting work. It is kept distinct from the *View* component which provides methods for the representation of the data structures maintained by the model on a presentation device (e.g. a visual or audio display). The *Controller* provides an interface from the interaction devices used by the user (e.g. the mouse, keyboard) and sends messages to the model to change its state or to the view to change the representation of the model.
Second, and more important for the environments we are considering here; this design facilitates the distribution of functional components over specialized computing resources. If designed and implemented properly, such a distributed system could provide a more responsive interactive system as compared to the situation where all components execute on the same resource. The separation of an interactive application into components requires some form of communication for the exchange of messages. The implementation of this message exchange mechanism raises additional concerns when the components are distributed over different systems.

5.2 High performance interactive simulation

The responsiveness of an interactive system is directly related to the rate at which updates are generated by each of the components in the system. To increase responsiveness, the delays between the consecutive components in the interactive system should be minimized. In an ideal system, each component produces results the moment input data is available and communication between components is instantaneous. In practice, however, there will always be some delay. The accumulation of all delays is referred to as "update time". In an interactive system there will always

be some delay from the moment interaction takes place until the moment that the environment has reacted to this interaction. This delay is referred to as “response time”.

5.2.1 Update and response time

In a non-interactive environment, the update time T_U is the sum over the execution time for the different components (T_{sim} for simulation, T_{vis} for visualization and T_{ren} for rendering) and the communication delay between components ($T_{sim\rightarrow vis}$ and $T_{vis\rightarrow ren}$):

$$T_U = T_{sim} + T_{sim\rightarrow vis} + T_{vis} + T_{vis\rightarrow ren} + T_{ren}. \tag{5.1}$$

Decreasing update time means that the delays imposed by the different components must be minimized. In the case of executing components this means that the time between the acceptance of input data and the production of results should be minimized. In the case of communication between components, the dominating factor for delay is the time that is required to transfer data from one component to the next.
In an interactive system, the response time T_R depends on which component the interaction is directed to, since only this and subsequent components need to be updated. In case the user interacts with the simulation component, the response time will be $T_R = T_{i(sim)} + T_U$, where $T_{i(sim)}$ is the delay between the moment an interaction with the simulation component was initiated and the moment it is received by the simulation.

5.2.2 Pipelined execution

A dynamic system differs from a static system in that the simulation component is an iterative process that repeatedly produces (intermediate) results. Basically, the delay at which these intermediate results become available is given by equation 5.1. Figure 5.2 shows a time-frame diagram in a “lock-step” IDEE. In this strategy, the simulation is allowed to advance only if the user explicitly tells the environment it is alright to do so. In this case we say the exploration system is “user driven”. While the user is exploring the results rendered by the graphics system, the simulation and visualization modules sit idle. In situations where a single simulation, a single visualization and a single rendering time-frame takes a negligible amount of time, this strategy may be perfectly adequate since the user will see the result of his interaction in short notice. However, if these time-frames are long, it may take a long time before the result of an interaction is shown. This can lead to an unusable environment and frustration with the user.
If the simulation component is (or can be considered as) a component that is independent of the execution of subsequent components, the update time of the whole environment can benefit from a pipelined execution model. In this execution model, a component resumes execution as soon as its output data has been accepted by the next. The time required before simulation updates are presented to the user (i.e. the length of a time frame on the exploration level) is shortened by allowing the simulation to run in parallel to the rest of the environment, as illustrated by the time-frame

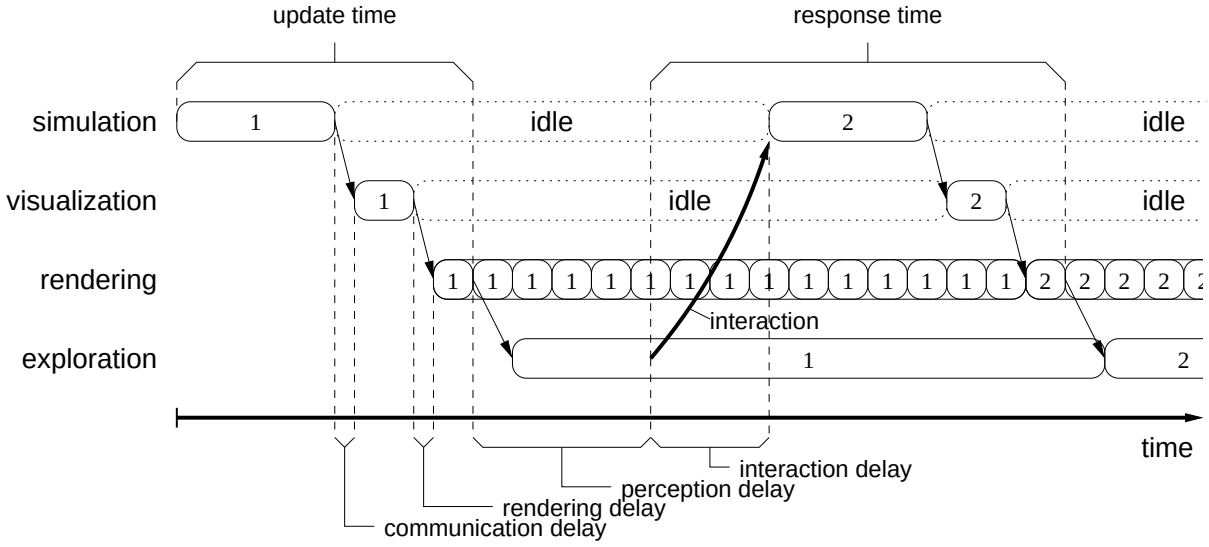


Figure 5.2: Time-frames and delays in a lock-step, user driven interactive dynamic exploration environment (IDEE). Time-frames are illustrated by rounded boxes. The gaps between time-frames on a same level represent the idle time of the component on that level. The gaps between time-frames on neighbouring levels represent the delays that occur between the time one component is done with a time-frame and the next component starts working on it. These delays are delineated at the bottom of the figure.

diagram in Figure 5.3. In this case simulation, visualization and rendering execute in parallel. Once all components in the pipeline have executed at least once, and provided sufficient resources are available to execute all components simultaneously, the update time becomes

$$T_U = max(T_{sim} + T_{sim \rightarrow vis}, T_{vis} + T_{vis \rightarrow ren}, T_{ren}). \tag{5.2}$$

The frequency of updates perceived by the user at the exploration level depends on the component that requires the longest amount of time to process one time-step. In the case illustrated in Figure 5.3, the simulation is the longest executing component in which case we say that the system as a whole is “simulation driven”.

As the components in this execution model are allowed to proceed after having processed one time-step, a situation results where after some time each component is processing a different time-step. Specifically, the user at the exploration level looks at information that was calculated by the simulation component “in the past”. In case the user performs no interaction with the data this is perfectly fine. However, if the user *does* interact with the data and if this interaction has a direct influence on earlier components (such as the simulation), the component with which the interaction took place may have to rollback to the time step with which the interaction took place in order to obtain consistent results. We will come back to the consequences of this situation when we discuss time management in section 5.3.4.

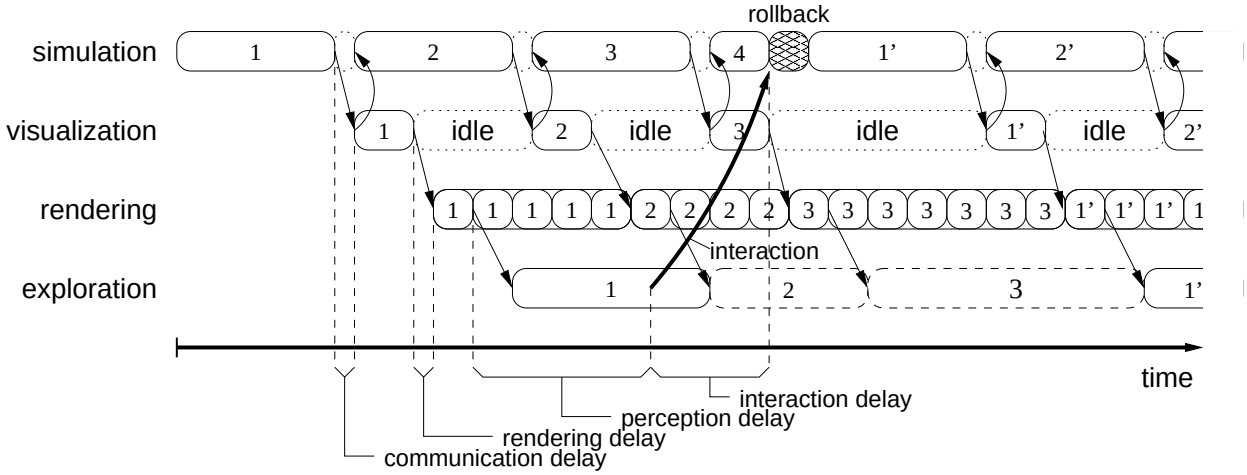


Figure 5.3: Time-frames and delays in a pipelined, simulation driven IDEE.

5.3 Distributed simulation and visualization

The capabilities of modern computer systems may, in some cases, allow both the simulation and visualization components to be performed on the same machine. However, a performance increase may be attained by running these components on dedicated computing platforms. For example, many simulation applications perform better on dedicated hardware such as vector processors, massively parallel platforms or other high performance computing machinery. State-of-the-art graphical systems are now available that are well suited for the rendering tasks. Moreover, a decomposition of an IDEE into separate communicating components facilitates implementation and allows more control over the performance of the system as a whole. However, the decomposition of an IDEE over distributed systems has a number of implications that need to be addressed to obtain a usable exploration system, as described next.

5.3.1 Execution environment

Especially in the case of distributed environments, some means of job control is needed that allocates the resources required for the application prior to execution (when, for example, execution needs to take place on batch queue execution systems). In many organizations such an environment will need to adhere to on-site authentication and authorization regulations as stipulated by the local organization. Furthermore, a mechanism should be provided that allows the user to start/stop the execution of the different components of the environment on the various computing platforms with minimal effort. The application developer cannot expect that an executable and a startup script handed to a user can be successfully used to deploy this program, especially not when distributed execution is involved. Failure is almost certain and it can take days to track down and fix problems. “Globus” is one example of a software infrastructure for distributed computation that integrates geographically distributed computational and information resources [81]. Globus provides a “single sign-on”

service that requires a user to be authenticated only once. After this, the user may execute jobs on any remote system for which they have been given authorization.

5.3.2 Data distribution

In distributed systems, components execute on different, possibly heterogeneous computing platforms. To be able to communicate data with each other, components provide access to their attributes, which can then be made available to other components. In heterogeneous computing environments, the attributes often have to be converted into different representation formats. Furthermore, in many circumstances not all components in an environment will participate in a communication. For these situations a publication and subscription mechanism needs to be provided that limits communication to members of a restricted group. Message passing systems such as PVM and MPI support this data distribution facility for the most part [80, 229]. In general, however, these systems do not support the construction of asynchronous systems as they restrict the application programmer to the Single Program Multiple Data (SPMD) communication paradigm, which in an IDEE would be too restrictive.
In most of the interactive simulation systems we are interested in, the data volumes that are generated by the simulation in each time-step are in the order of tens to hundreds of megabytes per time-step. For this reason we are particularly concerned with the communication performance provided by a data distribution facility in the case of large data volumes.

5.3.3 Attribute ownership

The behaviour of individual components in the environment is defined by one or more attributes (or parameters) which together define the state of that component. To avoid race conditions in a distributed system, attribute changes (which can be considered events, for example as a result of user interaction) should only be performed by a component that *owns* the attribute. In some cases it may be necessary to transfer ownership so that attributes can be changed by other components (for example in a collaborative environment where multiple users manipulate the same components).

5.3.4 Time management

An important issue in an IDEE is time management. In some situations it may be appropriate to constrain the progress of one component based on the progress of another. Time management deals with the exchange of time stamped information between components. For an IDEE, the four most time demanding components are; the simulation system, the visualization modules, the rendering layer and the explorer (i.e. the user).
Please recall Figure 5.3 in which a time-frame diagram is shown for a pipelined environment. In a pipelined system, each component is allowed to advance to the next

time-frame when it has finished processing and communicating the results of the current time-frame. Different components may therefore be processing different time-frames at the same wall-clock time. In addition, the rate of advance in time-frames is mostly irregular because of hardware, software and human imposed delays. As a consequence, time delays occur when the output generated by one component cannot be accepted for processing by another component immediately. When components depend on the output of an increasing number of other components, the difference between time-frames that is processed by components further apart in the pipeline increases. This has a causality consequence for the user who explores the final component in the pipeline and therefore interacts with presentations derived from a much "earlier" time-frame than what is being processed by the simulation at the same wall-clock time. Time management is responsible for preventing, or otherwise, detecting and resolving this causality violation. Methods for resolving time causality problems have been investigated within our group [176].

5.4 Communication architectures

Interactive distributed simulation environments consist of interconnected communicating components. The performance of such a system is determined by the execution time of the executing components and the amount of data that is exchanged between components. In the following sections, we will look into a number of different architectures that may be suitable for the implementation of IDEEs. We will describe the capabilities of these systems for the implementation of the model depicted in Figure 1.2 (page 4). As noted earlier; most of the interactive simulation systems that interest us generate data volumes that are in the order of tens to hundreds of megabytes per time-step. For this reason, the communication performance of these systems for large data volumes will be analysed in more detail.

5.4.1 The High Level Architecture

The High Level Architecture (HLA) is a technical framework for modeling and simulation. HLA aims to establish a common architecture for simulation to facilitate interoperability among simulations and promote the reuse of simulations and their components [26, 60–64, 165]. A successor to the DIS (Distributed Interactive Simulation) protocol, HLA provides a robust architecture with which distributed discrete event and other types of simulations can be designed. HLA has replaced DIS as the standard technical architecture for all United States Department of Defense (DoD) simulations since 1996 and has been accepted as an IEEE standard (IEEE1516) in the year 2000.
An HLA simulation consists of a *federation* that is composed of one or more *federates* that exchange information in the form of *objects* and *interactions* (see Figure 5.4). Interaction between federations is controlled by a "Run-Time Infrastructure" (RTI). HLA systems comprise *rules* which govern how federates and federations are

constructed (see Figure 5.5); an *interface specification* which governs how federates interact with the RTI; and an Object Model Template which provides a method for documenting key information about simulations and federations.

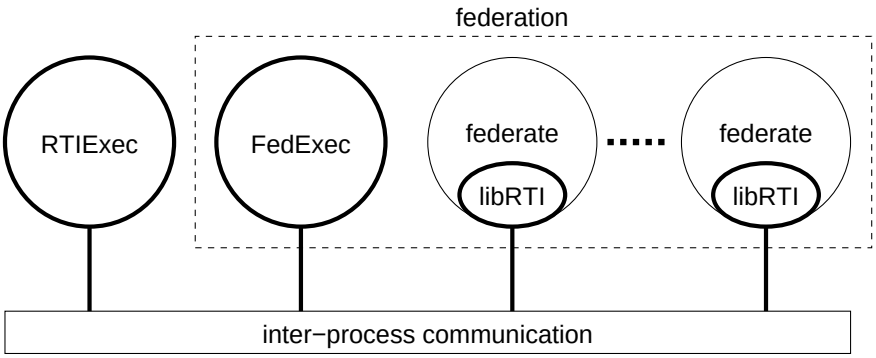


Figure 5.4: Components in HLA; federates combine to form a federation which is managed by a Federation Executive (FedExec) process. Inter-process communication is performed through services offered by the Run-Time Infrastructure Executive (RTIExec).

HLA provides solutions to many of the problems and issues described in section 5.3. Specifically, HLA allows data distribution across heterogeneous computing platforms (including message groups), supports a flexible attribute publish/subscribe and ownership mechanism and offers several methods to do time management.
The management services provided by the RTI are separated into six groups of functionality:

1. Federation management - this includes services for the creation of federations, joining federates to federations, observing synchronization points, federation-wide save and restores, resigning federates from federations and destroying federations.

2. Time management - this includes services that implement the policies and negotiations for advancing logical time. A federate may be "regulating" (meaning it is allowed to send time-stamped events), "constrained" (meaning it is capable of handling time-stamped events), "neither regulating nor constrained", or "both regulating and constrained". With each federate adopting one of these four policies, continuous time simulations as well as conservative and optimistic discrete event simulations can be constructed, or combinations of these within the same federation, if so desired.

3. Declaration management - this includes functionality for the publication, subscription of object instances or attributes.

4. Object management - these include the functions for the registration and instance updates at the object production side and instance discovery and reflection on the consumer side. Object management also includes methods for sending and receiving interactions, controlling instance updates based on consumer demand.

Federation Rules:

1. Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).
2. In a federation, all representation of objects in the FOM shall be in the federates, not in the run-time infrastructure (RTI).
3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
4. During a federation execution, federates shall interact with the run-time infrastructure (RTI) in accordance with the HLA interface specification.
5. During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

Federate Rules:

6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).
7. Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.
8. Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM.
9. Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.
10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

Figure 5.5: Governing rules for federations and federates in HLA.

5. Ownership management - controls which federates are allowed to change (attributes of) object instances.
6. Data distribution management (DDM) - the RTI acts as an “intelligent switch”, matching up producers and consumers of data based on their declared interest. DDM provides a way to further isolate publication and subscription interests by defining a “routing space”, defined by regions.

From the considerations described in the previous sections, the High Level Architecture (HLA) seems to be a suitable architecture for the construction of an IDEE. Figure 5.6 illustrates how an IDEE can be implemented using HLA. This IDEE consists of a simulation federate executing on a High Performance Computing (HPC) system which allows the simulation to run at best performance, and a visualization

and interaction federate that both run on a VR system. All federates are joined into one federation which is governed by a FedExec process. This FedExec process is created by the RTIExec process (which needs to be started manually) the moment the first federate is created and if the federation does not already exist. If the federation already exists, federates simply join the federation.
The federates in the federation communicate using the services provided by the RTI. In this IDEE, the simulation federate publishes simulation results using HLA's declaration and object management services. The visualization federate subscribes to the simulation results so that these are reflected as soon as results are available from the simulation. Interaction with both the visualization and the simulation federate is performed using HLA's object management services. Concurrent access to the simulation results is controlled by HLA's ownership management services while the time management services can be used to control time-frame advances. Both lock-step and pipelined execution models can be achieved with this IDEE through HLA's time management services; by defining the interaction federate as "regulating" and the simulation as "constrained", a lock-step execution model is obtained as illustrated in Figure 5.2. By defining the visualization federate as "regulating" and the simulation as "constrained", a pipelined execution model is obtained as illustrated in Figure 5.3.

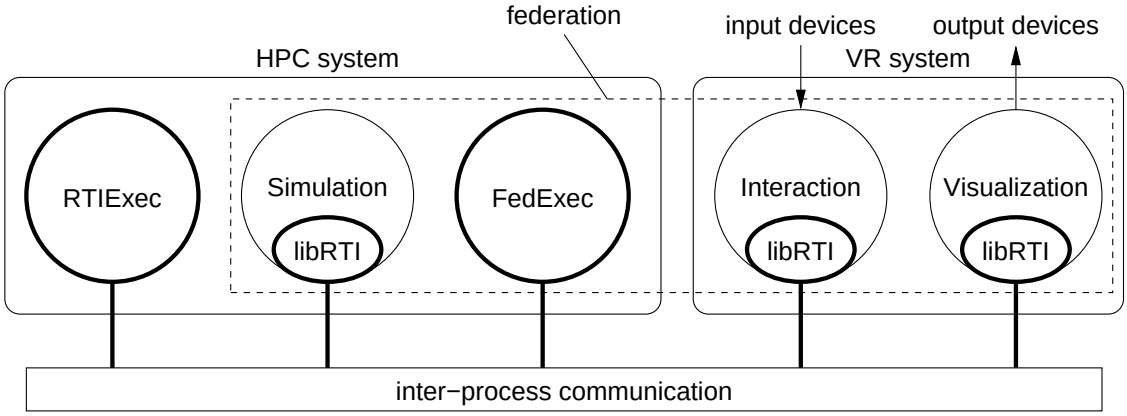


Figure 5.6: Structure of an IDEE using HLA; in this case, a simulation federate is running on a High Performance Computing (HPC) system, a visualization and interaction federate on a VR system. All take part in one single HLA federation, governed by a FedExec process. Communication between federates in the federation is performed through services offered by an RTIExec process.

Communication performance

Our primary concerns with HLA are with the performance of communication in HLA and the engineering effort imposed on developers. We have measured the performance of HLA by measuring the transfer time of various sized data blocks between two workstations. Both workstations are based on Pentium III processors, running Linux (RedHat 7.2) and connected via a shared 100 Mbit/s network, using both RTI

1.3NG version 3 and version 5 developed by the Defense Modeling and Simulation Office (DMSO) [60]. Details on the software architecture used for these measurements are described by Zhao et al. [269]. Each measurement in Figure 5.7 is the average over 15 samples. The deviation in the measurements is around one percent.

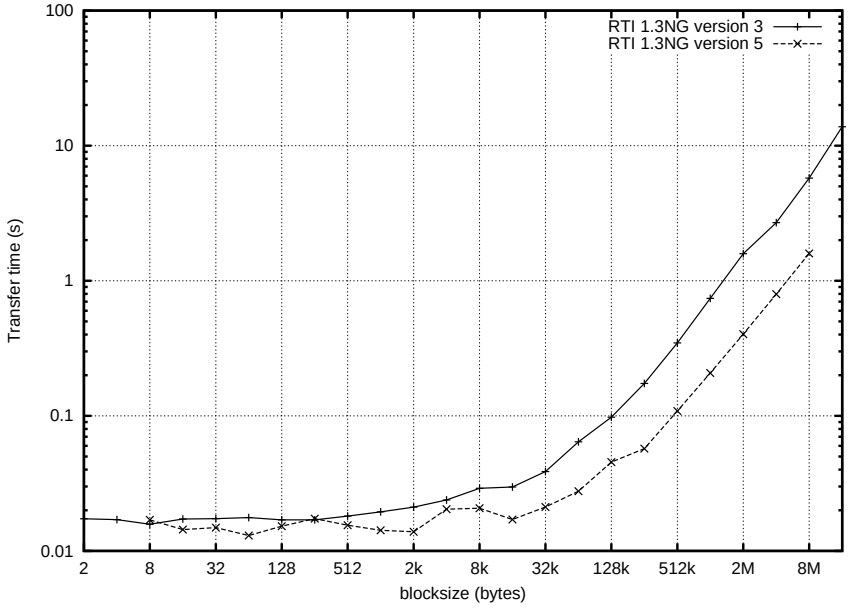


Figure 5.7: HLA communication performance between two Pentium III based Linux systems, connected by a 100 Mbit/s network. All measurements are averaged over 15 samples. The deviation in the samples is around one percent.

Internally, RTI uses TCP/IP sockets as its communication mechanism and we can see that the additional services provided by RTI do not put a severe penalty on communication, except for an additional latency for small messages of around 0.02 seconds. For packages sizes less than 512 bytes, we can see that the transfer time is more or less constant. It is very likely that the designers of this RTI implementation decided to optimize the communication of small sized messages, which is probably quite common in the type of simulation systems used by the Department of Defense (i.e. combat simulations). From package sizes of 64 kB and up we can see that the transfer time grows linearly and stabilizes at a throughput of around 1.2 Mbyte/s for RTI version 3 and 5.0 Mbyte/s for version 5. The increase in performance from version 3 to version 5 is significant. Little is known about the communication layer used in the RTI implementation provided by DMSO, but as they were involved with the design and development of CORBA, it could well be that the RTI makes use of "TAO", a real-time distributed Object Request Broker used in CORBA [270] [207, 248].

Engineering an HLA application

All objects and interactions between objects in a federation must be described according to an "Object Model Template" (OMT). This prerequisite forces the application

engineer to document his code through a formalized framework prior to (and during) implementation. By doing so, applications developed in the HLA framework can be automatically checked for correctness during development and at execution time. The OMT is used to document objects and interactions in a federation in a "Federation Object Model" (FOM), federates are documented in a "Simulation Object Model" (SOM), objects and interactions that are used to manage a federation are documented in a "Management Object Model" (MOM). The development of a SOM (for each federate in a federation) and a FOM (for each federation) is a requirement put forth in the "Federation Development and Execution Process" (FEDEP) model. To help the developer in the formulation of SOMs and FOMs, the "Object Model Development Tool" (OMDT) was developed by DMSO.
The engineering enforced by the FEDEP requires a substantial investment from the application developer. Although this design model forces the engineer to formally describe his code, which is intended to increase the quality of the end product, the initial investment, in terms of time, for the development of new federates is high. However; once defined, SOMs allow simulation components to be reused, which significantly reduces development time.

Conclusions

HLA is a complex but complete framework for the development of interactive distributed simulations. HLA provides most of the services described in section 5.3 that are needed for the construction of interactive distributed exploration systems. DMSO's RTI provides no execution environment that allows HLA simulations to be bootstrapped autonomously; each host participating in a federation requires that an RTI execution environment is started manually. Work is currently underway in our department to see whether HLA can be used in ways that avoid this. Although early version of DMSO's RTI showed communication performance results that were marginal, the performance of recent versions has increased significantly. The latest (and final) version of DMSO's RTI (version 6) was released on September 30, 2002 [67].

5.4.2 SPLICE

SPLICE is a software architecture developed at Hollandse Signaalapparaten B.V. (HSA) for large-scale distributed embedded systems [17, 18, 59]. Included in the design goals of SPLICE was the aim to provide an architecture that reduces the complexity of the development of large, reactive distributed systems, to provide fault-tolerance and to allow incremental deployment and development of systems.
SPLICE uses a data-oriented coordination model in which multiple processes exchange messages. Each process has its own communication "agent" that maintains a "local store" (see Figure 5.8). As in HLA, there is no explicit communication between processes. Instead, each process executes autonomously and the communication agents manage communication between distributed processes based on a subscription paradigm (hence the acronym; *Subscription Paradigm for the Logical Interconnection*

of Concurrent Engines). Each process publishes and/or subscribes to one or more "data sorts" that identify messages. When a process publishes a datum, its agent will store it in the local data store and forward the datum to all agents that manage a process that has subscribed to this datum. The data space that is thus created resembles the "tuple space" paradigm used in Linda [214]. As all processes work independently from each other, global control is minimized which greatly reduces the complexity of designing a distributed reactive system.

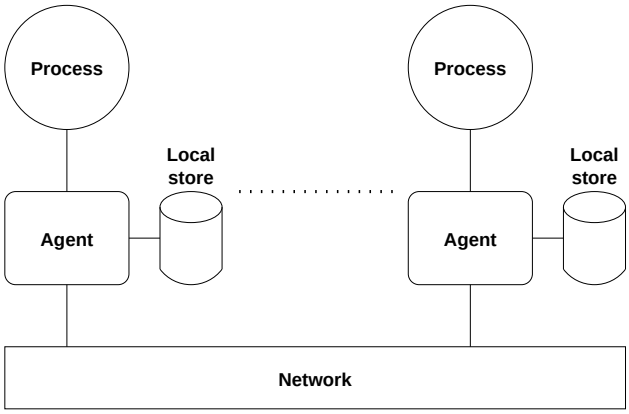


Figure 5.8: Components in SPLICE*: processes, each with an agent and a local store that communicate over a network.*

SPLICE makes a distinction between *periodic*, *context* and *persistent* data to handle different communication needs between processes. Periodic data sorts are used in situations where data is generated repetitively and the validity of this data decreases over time. Data in this category is delivered only to processes that are active at the time it is published. Context data sorts are used to represent the current state of the system as a whole. SPLICE ensures that the most up-to-date values are always available and provides all context data to each newly created processes. Persistent data is like context data except that storage is non-volatile. Sorts that are labeled "persistent" are stored in a persistent database even after processes have gone offline. When new processes require this data, the values are retrieved from the database.
Processes may identify fields in the data structure of a data sort as "key-fields". New instances of a dataset that are identical in all key-fields replace earlier produced data while all other instances will be stored separately. Even if the data structure in a dataset is the same for both producers and consumers, the key-fields may be different. This mechanism allows subscribers to index data that was produced while it was unable to consume the data.
Figure 5.9 illustrates how an IDEE can be implemented using SPLICE. This IDEE consists of a simulation process executing on a High Performance Computing (HPC) system which allows the simulation to run at best performance, and a visualization and interaction process that both run on a VR system. All processes communicate through local communication agents provided by SPLICE. In this IDEE, the simulation process publishes simulation results as "periodic" data through its local agent.

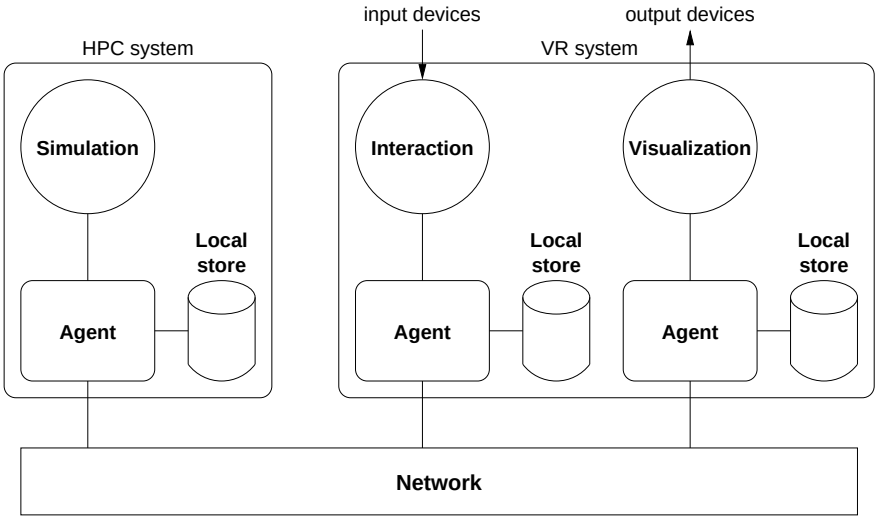


Figure 5.9: Structure of an IDEE using SPLICE*; in this case, a simulation process is running on a High Performance Computing (HPC) system, a visualization and interaction process on a VR system. Communication between processes is performed through local communication agents provided by* SPLICE*.*

The visualization process subscribes to the simulation results so that these are reflected by its local agent as soon as results are available from the simulation. By providing different "key-fields", separate time-frames of simulation results are maintained to avoid that simulation data is missed by the visualization or interaction processes in case they were unable to consume the data. Interaction with both the visualization and the simulation processes is performed using the same publication and subscription methods. Although SPLICE provides no explicit services to support ownership management and time management, the existing services can, in principle, be used to implement these.

Communication performance

As SPLICE is primarily designed for the exchange of small messages between many processes executing at the same time, we were concerned with its performance for the communication of large data volumes. To determine this, we designed and implemented a program using SPLICE that exchanges large messages between two distributed systems[†]. Figure 5.10 shows the communication performance of SPLICE (version 3.8) measured between two Sun UltraSparcs, connected by a 100 Mbit/s network connection. Each measurement is averaged over 15 samples.
The measurements show that the increase in transfer time differs below and over a message size of 1 Mbyte. At this message size, the transfer rate is approximately 500 kByte/s. For larger messages, the transfer time increases at a steeper rate and trans-

[†]Many thanks to Erik Boasson, HSA B.V., for providing us with a version of SPLICE for these experiments and for his support in answering questions.

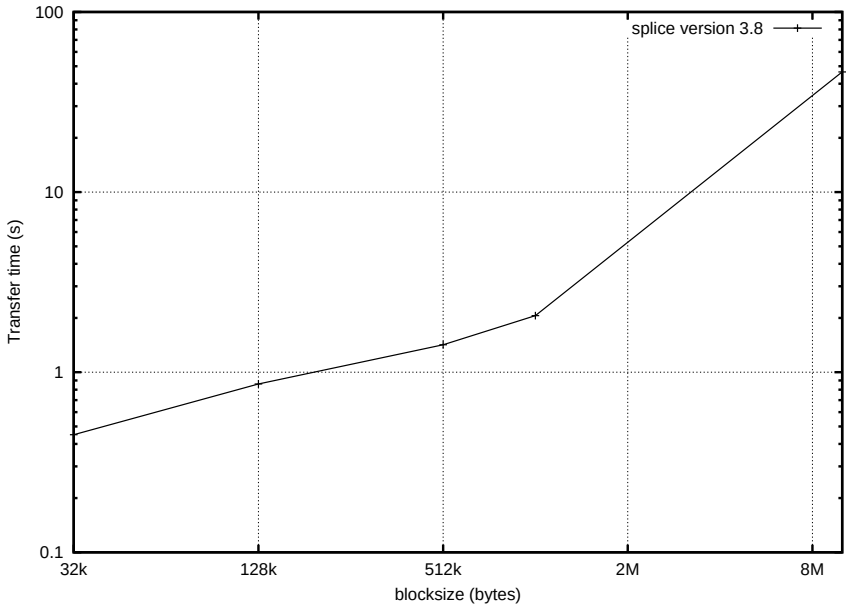


Figure 5.10: SPLICE communication performance for large messages between two Sun connected by a 100 Mbit/s network connection. Each measurement is averaged over 15 samples.

fer rate drops back to approximately 200 kByte/s for a 10 Mbyte message. It turns out that this is related directly to the amount of shared memory available on a system. SPLICE stores data into shared memory to improve communication between processes on the same system. However, it also uses this memory as a buffer for communication to processes on other systems by the communication agents. Unfortunately, shared memory is a limited resource on most operating systems. When the amount of shared memory is depleted, SPLICE is forced to break up messages into smaller blocks. This, as well as the overhead imposed by the exchange of multiple blocks for one message, reduces the communication throughput for large messages.

Conclusions

The paradigm provided by SPLICE allows for the development of large, complex distributed systems with relative ease. It is clear that HLA and SPLICE share many of the same design decisions, however, the construction of distributed systems using SPLICE is much easier. SPLICE provides most of the services described in section 5.3 that are needed for the construction of interactive distributed exploration systems. However, the implementation we used for our performance measurements showed that SPLICE is less suitable for systems that exchange large data volumes.

5.4.3 The Virtual Laboratory and the Data Grid

VLAM-G, the Grid-based Virtual Laboratory AMsterdam (or VL for short), provides a science portal for distributed analysis in applied scientific research. It offers scientists the possibility to carry out their experiments in a familiar environment, where content and data are clearly separated. Emphasis is put on the development and use of open standards and seamless integration of external computational resources across organizational borders [3]. A modular architecture is essential to establish such a flexibility. Supporting inter-disciplinary interactions implies the composition of (software) modules, which may have been developed independently. VLAM-G takes this modularity into account by providing a modular data flow-based system. In order to ensure that only proper connections can be established, the constituting modules communicate using a strongly typed communication mechanism.

A common Run-Time System (RTS) has been designed which can properly deal with a set of generic modules for various scientific areas. A preliminary requirement analysis performed together with various scientists from the three application domains has led to a classification of three types of modules: control modules that control external devices, processing modules that perform data filtering and visualization, and database and mass storage modules that allow the experimenter to store and retrieve local and remote data, either in raw data files or in databases. A scientist assembles these modules to compose his experiments. VL middleware assists the scientist by providing a Graphical User Interface (GUI) and an assistant. These components ease the handling of remote data access, resource allocation, security issues and access to external devices. The editing phase of an experiment is performed using a portal, implemented as a drag-and-drop GUI.

Communication in VL

Modules in VL start execution when consumable data is available and produce results for other processes to consume, much in the same way as in visualization pipelines described earlier in section 1.3.2 (page 7) [262]. The main difference with SPLICE and HLA is that the communication path from one process to another is explicit in VL. Processes in VL are logically connected in Process Flow Templates (PFT) that identify (1) the modules that are part of an application (a so-called "experiment" in VL terminology) and (2) the logical connections between the modules (see also Figure 5.11). So-called "portals" provide access to the different aspects of VL for the definition of experiments, launching experiments, data storage of intermediate results and visualization front ends for (intermediate) representation of experimental results. Communication between modules is performed by the Virtual Laboratory Abstract Machine (VLAM) which uses the Globus communication primitives to transfer data across organizational borders [81, 88].

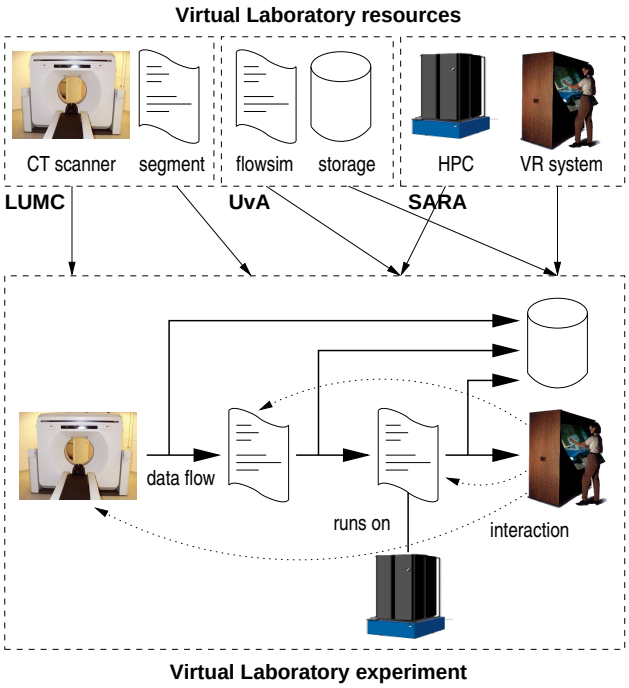


Figure 5.11: An example of an experiment definition in the Virtual Laboratory. This example shows an IDEE where resources from different institutes are combined into an experiment for interactive flow simulations through medical scans (see also Chapter 6).

Concluding remarks

The design of VL addresses some of the issues described in section 5.3 for the construction of an interactive dynamic exploration environment (IDEE). In particular, VL provides an execution environment that hides many of the details of the execution of distributed processes on different systems across organizational domains. The communication and execution mechanisms provided by VL are sufficient for the implementation of a pipelined distributed IDEE as described at the beginning of this chapter, although they do not provide the richness of capabilities offered by both HLA and SPLICE. Unfortunately, the performance of the communication mechanisms in current implementations of VL is inadequate for the exchange of large data volumes (data not shown).

5.4.4 Concluding remarks on communication architectures

The communication architectures just described address many, though not all, of the design criteria for the construction of an IDEE that were set forth in section 5.3. HLA (the High Level Architecture) provides solutions to many of the problems and issues described in the previous sections and is, therefore, a suitable architecture for constructing an IDEE. Specifically, HLA allows data distribution across heterogeneous computing platforms (including message groups), supports a flexible attribute

publish/subscribe and ownership mechanism and offers several methods to do time management. The use of HLA in interactive simulation systems is described in more detail by Zhao in [268, 269]; this work describes an Agent-based Intelligent Virtual environment (AG-IVE) where agents are used to coordinate communication between distributed systems on behalf of processes that take part in an interactive simulation system.
One concern that remains is that of the interchange of large data volumes in an IDEE. Again; as noted earlier, most of the interactive simulation systems that interest us generate data volumes that are in the order of tens to hundreds of megabytes per time-step. In the following section we describe a number of techniques that can be used to increase the throughput of a communication architecture. These mechanisms are used in a test case of an IDEE which will be described in chapter 6.

5.5 High throughput communication using CAVERN

Distributing components over different systems means that some form of communication must be established to allow the output of one component to be transferred to the next. It could be that the overhead generated by this communication mechanism annuls the benefits obtained by the distribution (i.e. although T_{c1} decreases through the use of optimized resources, $T_{c1 \rightarrow c2}$ increases because of the extra communication overhead between two components $c1$ and $c2$). To reduce the delays caused by communication it may be beneficial to reduce the amount of transferred data as much as possible. This reduction itself, however, also takes time so careful consideration is often necessary.
We have implemented three mechanisms to increase the throughput of data transfers over a network. These mechanisms are cascaded into a pipeline to decrease the amount of transferred data and spread the remaining data over multiple network connections, in an effort to maximize throughput (see Figure 5.12).

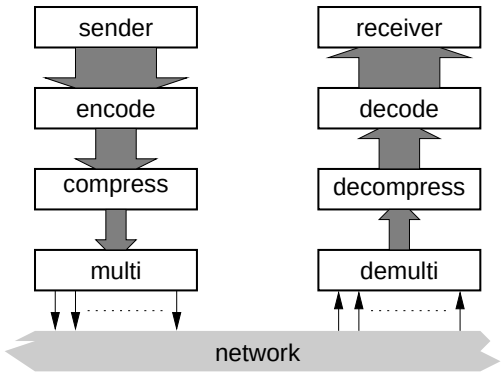


Figure 5.12: The stages in the communication pipeline: the data volume produced by a sender is reduced in volume by an application specific encoder and a compression stage. The remaining data is sent through multiple network connection to the receiver where the pipeline reconstructs the data for the receiver.

5.5.1 Hiding latency by using multiple connections.

The first method increases communication throughput by using multiple network connections between peers at the same time. There are two reasons why this increases throughput (see also Figure 5.13). First; in the case of reliable network connections (such as connections using the TCP protocol), delays caused by waiting for acknowledgment packets can be "hidden" by serving other connections that are ready. Second; the technique can exploit "intelligent" network devices that spread communication over different routes. This technique therefore performs best when there are many such devices between peers (which is often the case when peers are geographically dispersed). In principle, data can travel along different routes from peer-to-peer, thereby circumventing congestion caused by other traffic on the network.

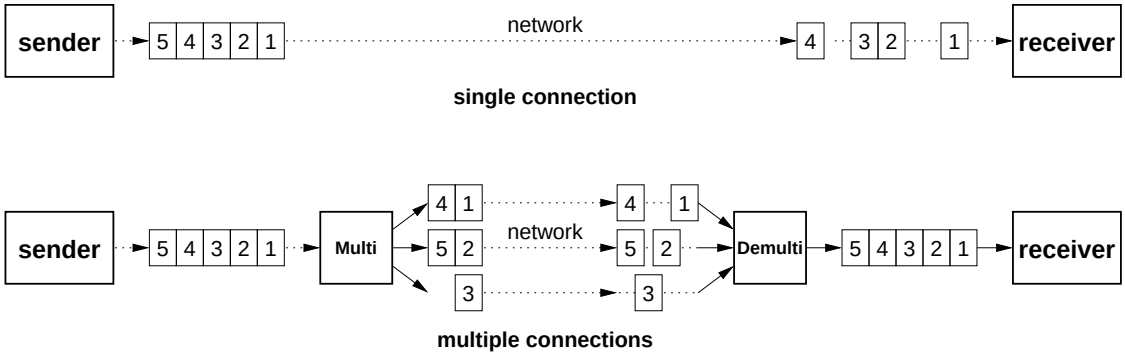


Figure 5.13: Increasing communication throughput using multiple network connections.

5.5.2 Data volume reduction

Note that the total volume of data that is transferred between peers is not decreased by the method described in the previous section. Instead, it increases throughput by exploiting as much available bandwidth as possible. The techniques described next aim to increase throughput by decreasing the volume of communicated data.

Data encoding

Data encoding is a semantic volume reduction technique that aims to reduce the volume that is needed to represent the data to a minimum. This technique relies on the fact that the receiving side may not always be interested in the most accurate representation of the data that was calculated by the sender. Because this type of data reduction throws away unnecessary information, this method is frequently referred to as *lossy compression*.
Although a significant reduction in volume can be achieved using this technique, the receiver should be conscious of the fact that the data it has received is not of the same accuracy as was originally produced. Although this reduction may be acceptable

under some circumstances, unexpected side effects may occur when the errors that are present in the data are accumulated due to an integrating method of analysis on the data. For example, an often used technique in vector field visualization is to represent the path of the flow using streamlines. These streamlines are created by integrating over individual vectors. Due to the accumulation of (small) errors in each individual vector, a streamline may follow a radically different path.

Data compression

Freely available compression libraries (such as *zlib* [152]) provide means of reducing data volumes very effectively. This data reduction does not make any assumption about the data contents and is therefore without any loss of information. Most compression libraries can be parameterized to indicate the level of compression that should be achieved at the expense of higher execution time. This type of data reduction is commonly referred to as *lossless compression* as the data is unchanged after decompression. The amount of compression depends on (1) the type of data and (2) the amount of data. In general, the compression ratio decreases when the amount of data decreases. Note that this is important in parallelized applications in which the original data volume is often decomposed into smaller subvolumes.

5.5.3 Performance of the network communication pipeline

We have implemented the mechanisms described above using CAVERN [145, 169]. In this section we show some results obtained from measurements on their performance.

Figure 5.14 shows the mean throughput over 200 measurements of the multiple connection stage in the communication pipeline. These measurements were taken between a system in Amsterdam and St. Petersburg, Russia. Over a single connection, the throughput between these systems was approximately 100 Kbyte/s. As can be seen from this figure, the average throughput increases as more connections are used, but up to a maximum. Using more connections congests the network and no further increase in throughput can be obtained. As more connections are used, the throughput becomes more and more unpredictable and decreases when too many simultaneous connections are used.

To illustrate the influence of encoding on throughput, Figure 5.15 shows the mean performance over 5 measurements of the complete network communication pipeline on the transfer of the results from a parallel lattice Boltzmann flow simulation kernel to a visualization frontend; using compression, multiple network connections, both with and without encoding. This figure illustrates that, on average, encoding doubles throughput. Although this figure shows the typical throughput that can be achieved, in some situations the total performance of the network communication pipeline resulted in a throughput of 62 Mbyte/s, which is over 5 times the bandwidth of the slowest network link (100 Mbit/s).

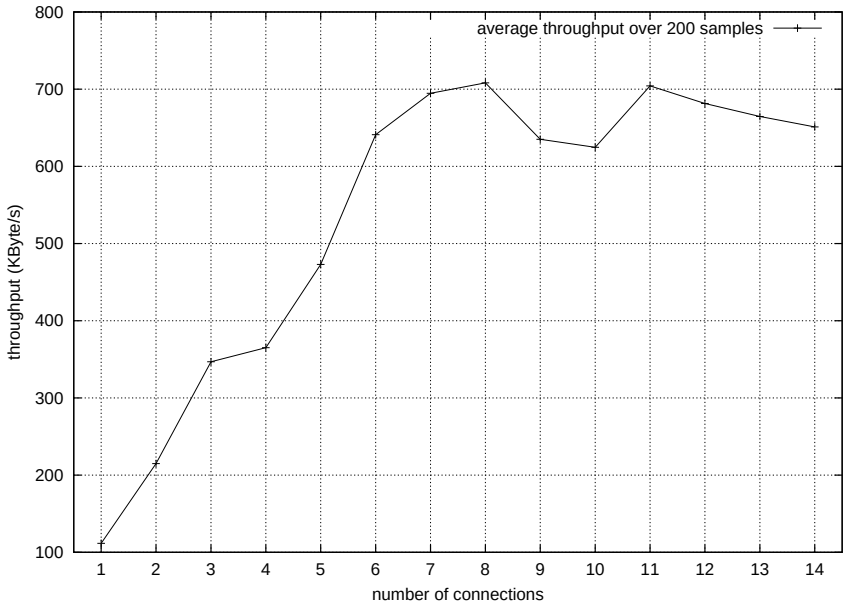


Figure 5.14: Average network throughput when using multiple network connections between a system in Amsterdam and a system in St. Petersburg, Russia.

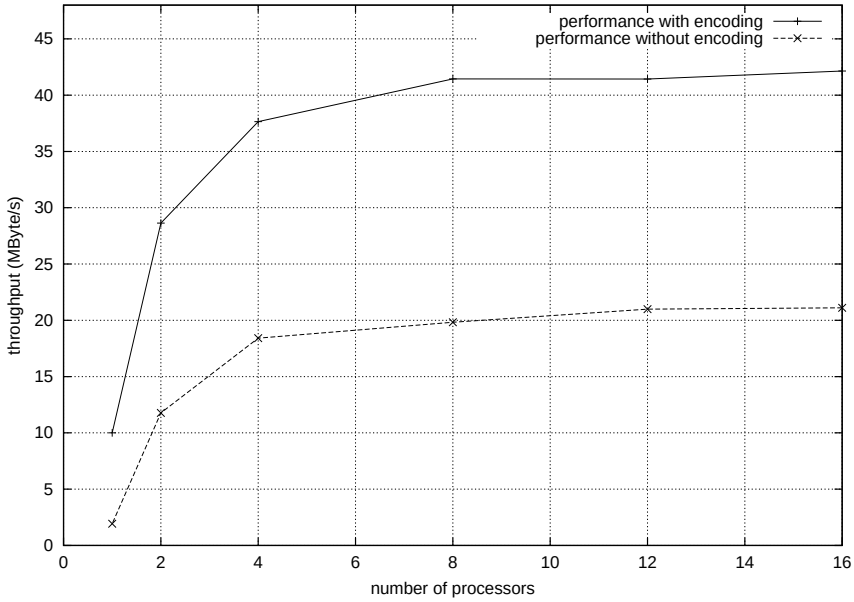


Figure 5.15: Mean throughput of the network communication pipeline when used to transfer the results from a parallel lattice Boltzmann flow simulation kernel to a visualization frontend; shown with and without encoding. Multiple network connections and data compression were used in both measurements.

5.5.4 Conclusions

Our measurements show that a great performance increase can be obtained with the mechanisms described in this section. The results presented here were obtained

using unoptimized algorithms; we expect that performance could be increased even more. The different stages in the communication pipeline require a certain amount of time to execute which adds delay to communication time. By tuning the parameters that influence this execution time, throughput can (in principle) be automatically optimized. For example; networks are generally shared by many institutes so that available bandwidth changes over time. The multiple connection technique can sense this change by measuring the effect of employing more or fewer connections during communication. An optimal number of connections can thus be determined dynamically (and transparently) while peers communicate. However, a different parameterization at one stage influences the execution time of subsequent stages which implies that parameter optimization is not a trivial task.

5.6 Summary and conclusions

This chapter addressed the issues involved in the design and implementation of high performance interactive dynamic exploration environments (IDEE). We argued that a design where processing, presentation and interaction are isolated into separate components can facilitate the engineering of an IDEE and allows the components to be distributed over specialized computing resources to increase performance.
We described two execution models for IDEEs; a user driven and a simulation driven execution model, and discussed the performance behaviour and functional implications in the use and implementation of these models. The decomposition of an IDEE over distributed computing resources requires a supporting communication architecture. Several communication architectures were described and evaluated; the High Level Architecture (HLA), SPLICE, and the Grid-based Virtual Laboratory AMsterdam (VLAM-G). We analysed the communication performance for the exchange of large data volumes in more detail.
The overall performance of a distributed IDEE benefits from high throughput communication. We designed three independent, cascadable mechanisms to increase communication throughput and measured their performance. These measurements show that a great increase in communication throughput can be obtained with these mechanisms. However, the measurements also suggest that throughput can be improved further by dynamically tuning the behaviour of the different stages at run-time.

Chapter 6

Integrating all: simulated vascular reconstruction in a virtual operating theatre*

"Training for minimal invasive surgery, or "trying to get the most gut for the cut", has its limitation due to the lack of repetitive training opportunities on either animals or humans."

Newsbytes News Network, 1997.

6.1 Introduction

In this chapter we describe an interactive dynamic exploration environment (IDEE) that combines simulation and interactive visualization in virtual reality (VR) into an environment that enables pre-operative surgical planning of vascular reconstruction procedures. This environment uses many of the ideas described in the previous chapters. It shows how the simulation and interactive visualization execute on distributed systems, communicating with each other over a high throughput network interface. Visualization of the simulation results and interaction with the environment take place from within a virtual environment (VE). The simulation runs on a parallel system for best performance.

6.1.1 Abdominal vascular reconstruction

Vascular disorders in general fall into two categories: *stenosis*, a constriction or narrowing of the artery by the buildup over time of fat, cholesterol and other substances in the vascular wall, and *aneurysms*, a ballooning-out of the wall of an artery, vein

*Parts of this chapter have been published in *R.G. Belleman and P.M.A. Sloot. "Simulated vascular reconstruction in a virtual operating theatre", number 1230 in* Excerpta Medica, International Congress Series, pages 938–944, 2001.

or the heart due to weakening of the wall [15, 34]. Aneurysms are often caused or aggravated by high blood pressure. They are not always life-threatening, but serious consequences can result if one bursts.
A vascular disorder can be detected by several imaging techniques such as X-ray angiography, MRI (magnetic resonance imaging) or computed tomography (CT). Magnetic resonance angiography (MRA) has excited the interest of many physicians working in cardiovascular disease because of its ability to non-invasively visualize vascular disease. Its potential to replace conventional X-ray angiography that uses iodinated contrast has been recognized for many years, and this interest has been stimulated by the current emphasis on cost containment, outpatient evaluation, and minimally invasive diagnosis and therapy [266].
A surgeon may decide on different treatments in different circumstances and on different occasions but all these treatments aim to improve the blood flow of the affected area . Common options include *thrombolysis* where a blood clot dissolving drug is injected into, or adjacent to, the affected area using a catheter; *balloon angioplasty* and *stent placement* which is used to widen a narrowed vessel by means of a inflatable balloon or supporting framework; or *vascular surgery*. A surgeon resorts to vascular surgery when less invasive treatments are unavailable. In *endarterectomy* the surgeon opens the artery to remove plaque buildup in the affected areas. In vascular bypass operations, the diseased artery is shunted using a graft or a healthy vein harvested from the arm or leg.
The purpose of vascular reconstruction is to redirect and augment blood flow, or perhaps repair a weakened or aneurysmal vessel through a surgical procedure. Although the optimal procedure is often obvious, this is not always the case; for example, in a patient with complicated or multi-level disease. Pre-operative surgical planning will allow evaluation of different procedures *a priori*, under various physiologic states such as rest and exercise, thereby increasing the chances of a positive outcome for the patient [234].

6.1.2 What is needed?

The test case described in section 6.1.1 contains all aspects of an interactive dynamic exploration environment. Our aim is to provide a surgeon with an environment in which he/she can try out a number of different bypass operations and see the influence of these bypasses. The environment needs the following:

- An environment that shows the patient under investigation with his affliction. A patient's medical scan is 3D, so to obtain best understanding on the nature of the problem, the surgeon should be able to look at his specific patient data in 3D, using unambiguous visualization methods. We have described the Virtual Radiology Explorer (VRE) environment that allows for the visualization of 3D medical scans in section 2.3 (page 29).

- An environment that allows the surgeon to *plan* a surgical procedure. Again, this environment should allow interaction in a 3D environment. For example,

the CAVE environment allows us to interact with 3D computer generated images using 6 degrees of freedom (DOF) interaction devices [52, 205]. We have described an architecture that allows interactive manipulation of virtual 3D objects in a virtual environment in section 4.2 (page 71). Note that *visual* realism is not the primary goal here; what is more important here is *physical* realism, and then only of particular features of fluid flow, as discussed later[†].

- An environment that shows the surgeon the effect of his planned surgical procedure. As the aim of the procedure is to improve the blood flow to the affected area, the surgeon must have some means to compare the flow of blood before and after the planned procedure. This requires the following:
  - A *simulation* environment that calculates the important properties of blood flowing through a patient's vascular system (i.e. pressure, velocity, wall shear stress). We will describe this simulation environment and the methods for obtaining the input data for this environment from patient specific data in section 6.2.
  - A *visualization* environment that presents the results of the simulation in a clear and unambiguous manner. We have described a scientific visualization environment that can be used in virtual environments in section 4.2 (page 71).
  - An *exploration* environment that allows the researcher to inspect and probe (qualitatively and quantitatively) the results of the simulation (e.g. means for performing measurements, annotate observations, releasing tracer particles in the blood stream, etc.). We have described an architecture that allows measurements to be taken in a virtual environment in section 4.5 (page 86). An environment that visualizes the results of blood flow simulation will be described in section 6.3.

6.2 The lattice Boltzmann method for flow simulation

The lattice Boltzmann method (LBM) is a mesoscopic approach for simulating fluid flow based on the kinetic Boltzmann equation. In this method fluid is modeled as particles moving on a regular lattice. At each time step particles propagate to neighboring lattice points and re-distribute their velocities in a local collision phase. This inherent spatial and temporal locality of the update rules makes this method ideal for parallel computing [123]. During recent years, LBM has been successfully used for simulating many complex fluid-dynamical problems, such as suspension flows, multi-phase flows, and fluid flow in porous media [134]. All these problems are quite difficult to simulate by conventional methods [122, 124].

[†]This in contrast to research projects toward virtual operating theatres that include the simulation of tissue deformation and realistic blood spills [12, 19].

The data structures required by LBM (cartesian grids) bear a great resemblance to the grids that come out of CT and MRI scanners. As a result, the amount of preprocessing can be kept to a minimum which reduces the risk of introducing errors due to data structure conversions. In addition, LBM has the benefit over other fluid flow simulation methods that flow around (or through) irregular geometries (like a vascular structure) can be simulated relatively simply. Yet another advantage of LBM is the possibility to calculate the shear stress on the arteries directly from the densities of the particle distributions [8]. This may be beneficial in cases where we want to interfere with the simulation while the velocity and the stress field are still developing, thus supporting fast data-updating given a proposed change in simulation parameters from the interaction modules.

6.2.1 Performance of the parallel LBM flow simulation kernel

Figure 6.1 illustrates the performance increase that is achieved by using a parallelized implementation of the flow simulation kernel on three different parallel computing systems;

- a 128 node SGI Origin 2000 system, each node consisting of a MIPS R12000 processor running at 300 MHz, running IRIX; the processors are interconnected through a ccNUMA (cache coherent Non Uniform Memory Access) architecture;
- a 20 node cluster-of-workstations (COW), each node consisting of 2 symmetric multi-processing (SMP) Intel Pentium-II processors running at 300 MHz (i.e. the system contains 40 processors in total), running Solaris, interconnected by Myrinet;
- a 40 node cluster-of-workstations (COW), each node consisting of an AMD Athlon processor running at 700 MHz, running Linux, interconnected with a dedicated fast (100 Mb/s) switched ethernet.

Figure 6.2 shows the speedup of the parallel implementation. Although the execution time per iteration was higher on the SGI system, this figure shows that the code runs at higher efficiency. This can be largely attributed to the more efficient communication architecture that is used in the SGI system. The figure also shows erratic speedup behaviour on the 20-node COW when more than 20 processes are used, even though the system has 40 processors. This can be explained from the fact that each node contains two processors but one network interface; when more than 20 processes are scheduled on the system, some nodes will run multiple processes on the same node, scheduled over the two available processors. When communication takes place, the processors compete for the single network interface, thereby inhibiting speedup.
Because the test case environment's architecture uses the distributed pipeline architecture described in section 5.2.2 and because the simulation component is, in general, the slowest executing component, the performance of the simulation environment in total is greatly increased. In the current LBM simulation environment the execution time of the lattice Boltzmann kernel is independent of the morphology through or

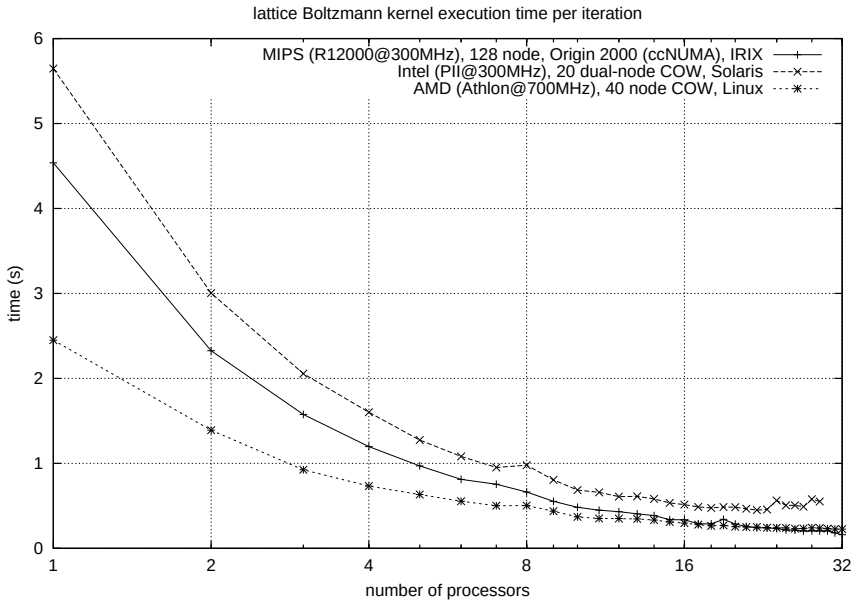


Figure 6.1: Iteration time of the parallel lattice Boltzmann kernel on different multiprocessor systems.

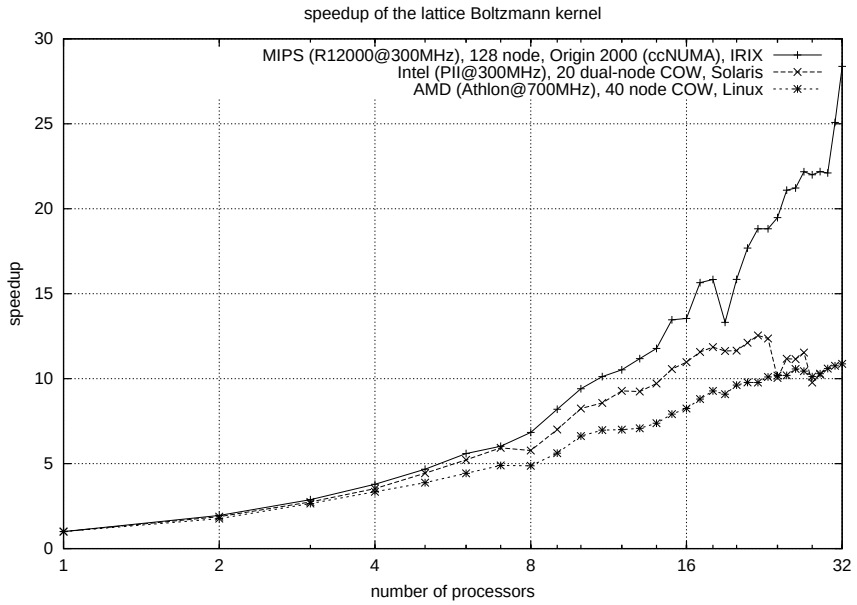


Figure 6.2: Speedup of the parallel lattice Boltzmann kernel on different multiprocessor systems.

around which the flow is calculated (given equal grid dimensions and simulation parameters). However, the ratio of fluid nodes and solid nodes can make an important difference. Especially in angio-vascular applications, the number of solid nodes is often much higher than the number of fluid nodes. An optimization can be applied in

the lattice Boltzmann kernel to speed up the simulation by disregarding solid nodes in the propagation phase.

6.2.2 LBM grid generation

For our purpose, we consider lattice Boltzmann grids to consist of isotropically structured nodes with one of the following five properties (see also Figure 6.3):

1. Fluid nodes define the geometry through which flow is simulated.
2. Solid nodes delimit the domain of fluid nodes, and as such define the "shape" of the geometry around (or through) which the fluid flow is simulated.
3. Boundary nodes are solid nodes that border fluid nodes. They are often defined as "first order bounce back nodes" or as "second order no-slip nodes" to describe the behaviour of the flow at the boundary of a solid and the fluid. What exactly constitutes a "border" depends on the type of neighbourhood that is used in the LBM algorithm; in 2D, the D2Q5 and D2Q9 models are often used (respectively with a 4 and 8 boundary neighbourhood); in 3D, the D3Q19 model is very popular (26 boundary neighbourhood).
4. Inlet nodes define the initial distribution of forces from where the flow enters the system. Often the inlet flow profile is statically defined to approximate a system in which the flow evolves under the influence of a time-continuous velocity profile. In dynamic systems, the inlet velocity profile is changed dynamically over time to approximate the forces that are exerted on the system by an external force (such as the pulsating flow as a result of a beating heart cycle, consisting of alternating cycles of *systoles* where blood is pumped out of the heart and *diastoles* where the heart is in rest).
5. Outlet nodes define the behaviour of the flow as it exits the system. Common conditions on the outlet nodes are "free flow", where the forces traveling out from the system encounter no resistance, "periodic flow", where the forces are wrapped back into the inlet nodes simulating a closed, recursive, system and "restrictive flow" in which a situation is approximated where flow encounters a high resistance (such as flow through small branches and capillaries that are too small to be accurately simulated using LBM).

LBM requires that the input grids comply to the following set of rules:

- inlet nodes may not neighbour boundary nodes, only fluid nodes,
- outlet nodes may not neighbour boundary nodes, only fluid nodes,
- solid nodes may only neighbour boundary nodes,
- boundary nodes may only neighbour fluid nodes.

The purpose of automatic LBM grid generation is to construct grids that comply to this set of rules.

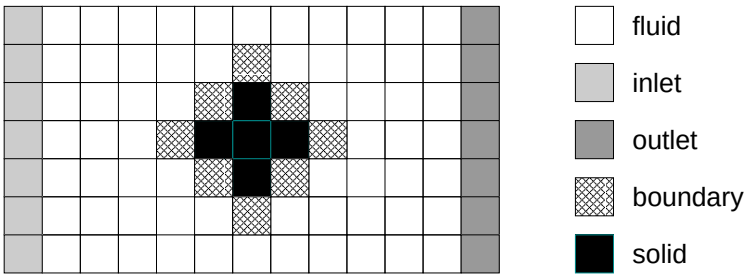


Figure 6.3: Different node types in an example D2Q5 lattice Boltzmann grid.

6.2.3 LBM grid generation from medical data sets

As mentioned earlier, the basic structure of the grids used in LBM bear great resemblance to the medical scans obtained from a patient by medical imaging devices such as CT and/or MRI scanners. In general these scanners produce stacks of two dimensional images that, when stacked together, form a three dimensional image volume.
For the purpose of flow simulation it is imperative that the structures through which the flow must be simulated are isolated from the raw medical scan as accurate as possible. This process is known as "image segmentation".

Image segmentation

Image segmentation, like flow simulation, is an area of active scientific research [9, 76, 148]. Many techniques exist, ranging from manual techniques where pixels are tagged by experienced radiologists with a thorough understanding of human anatomy, to fully automated techniques. All techniques rely on the basis that a sufficient signal-to-noise (S/N) ratio is present in the images so that the relationship of pixels with others that belong to the same structure can be identified. Obtaining images with a sufficiently high S/N ratio begins in the radiology department, at the time the scan is made.
In the case of vascular structures, the imaging process is referred as "angiography" (from the Greek *angeion*, "vessel" and *graphien*, "to write or record"). Radiologists are trained with the knowledge on the physics behind the different medical scanners and therefore know how to obtain good quality images. One currently used method is computerized tomographic angiography (CT angiography or CTA). Here, a high contrast CT scan is produced using intravenous water-soluble iodinated agents to image the vascular system. Time-of-flight magnetic resonance angiography (TOF-MRA) is a non-invasive imaging technique that is well suited for obtaining high S/N ratio scans. Although TOF-MRA does not require a contrast agent, it has the disadvantage that the acquisition times are usually high (in the range of 10-30 minutes for high resolution scans). This may hamper the maximum attainable resolution in regions of the body that move under the influence of respiratory motion.
Provided that sufficient contrast is present in the medical scans, the raw data from the medical scanner is first segmented so that only the arterial structures of inter-

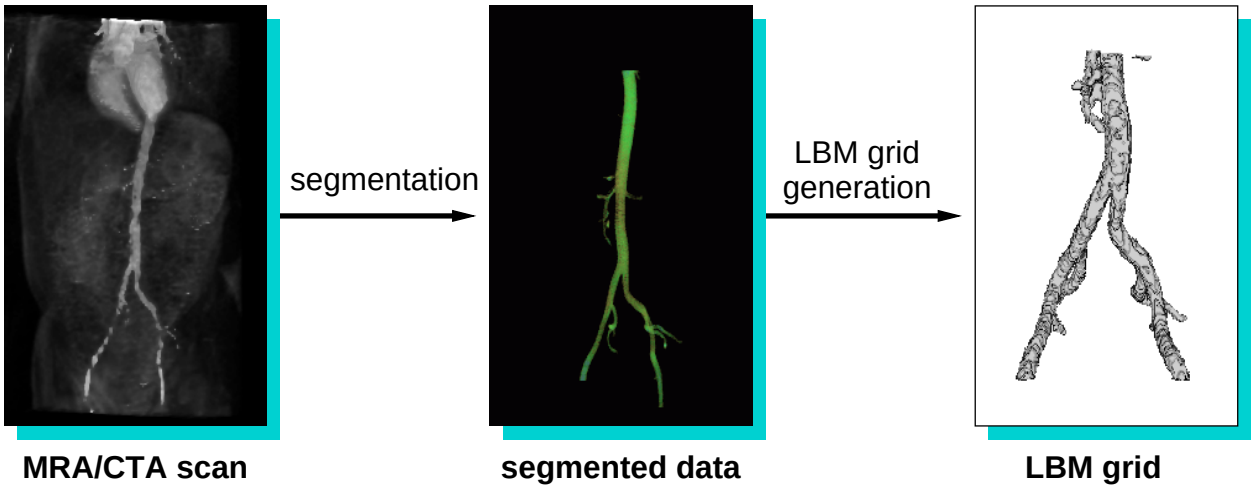


Figure 6.4: LBM grids are generated from raw medical scans through a combination of segmentation and image processing techniques.

est remain in the data set (see also Figure 6.4). In some rare cases, the S/N ratio may be sufficiently high that a level-threshold segmentation technique is suitable to segment the structure of interest from the raw medical scan. If this is not possible, more sophisticated techniques are required, such as for example the wave propagation technique developed at the Leiden University Medical Center [206].

Automatic generation of boundary nodes

The segmented data set is then converted into a grid that can be used in LBM; boundary nodes, inlet nodes and outlet nodes are added to the grid using a variety of image processing techniques. The implementation of the Lattice Boltzmann used requires that grids are isotropic. This means that the settings of the medical scanner either have to be such that isotropic voxels are produced or, otherwise, the image has to be resampled onto an isotropic grid. Conventional image processing techniques can be applied here, provided that valid interpolation algorithms are used as determined by the acquisition method that was used.
Boundary nodes are generated using a morphological image operation known as image dilation (see Figure 6.5). The effect of this operation is that one layer of nodes is generated around the segmented image that conforms to the requirements of the LBM simulation. By subtracting the original segmented image from the dilated image, an image is obtained that contains the boundary nodes only.

6.2.4 LBM grid generation and editing from polygonal data

As our aim is to provide physicians with an environment in which vascular reconstruction procedures can be simulated, methods should be available by which a reconstructive procedure can in some way be approximated by a LBM grid. We limit

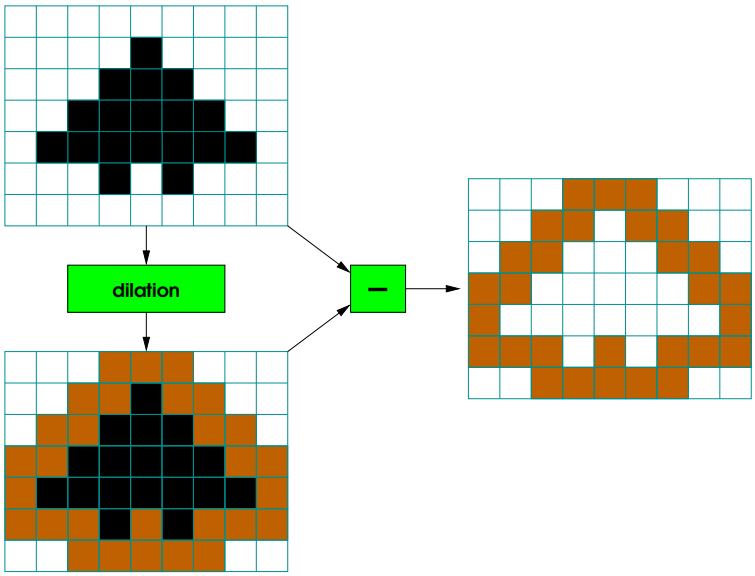


Figure 6.5: Boundary node generation using image dilation.

ourselves to reconstructive procedures that alter vascular geometry, more specifically, the types of treatment that are performed with stenotic and aneurysmal disease as described in section 6.1.1, with the exception of thrombolysis using chemical agents.

Distance sampling

A suitable method for generating grids from simple curves is distance sampling. Here, an implicit model is created from the input geometry by computing the smallest unsigned distance from the input geometry to each voxel in a grid. Figure 6.6 shows the execution pipeline that has been implemented to do this. The size of the output grid and the number of points in the input curve should be defined in such a way that a sufficiently accurate representation can be obtained.
The distance sampling algorithm visits each node in the output grid and records the

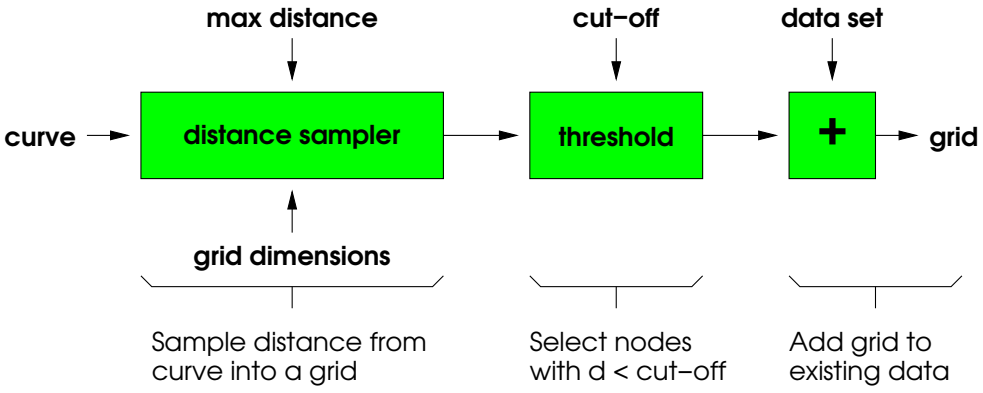


Figure 6.6: Grid generation using distance sampling.

Figure 6.7: Distance sampling from a line representation of the letter 'E' onto a 3-bit deep two dimensional image. Larger distance is represented by a darker colour.

minimal distance of this node to each of the points in the input curve. When this distance is represented by a colour, we obtain the picture shown in Figure 6.7. To decrease the execution time of the distance sampling algorithm, especially in the case of large grids, a maximum distance parameter limits the traversal to grid nodes that are within a specified distance from the input curve. The distances values stored in the sampled grid can be used to extract grid nodes with a distance less than a threshold value. This threshold value defines the diameter of the sampled curve; higher values result in wider structures while lower values in structures that are narrower. Finally, the thresholded grid is added to the medical data set for which the procedure it to be simulated.

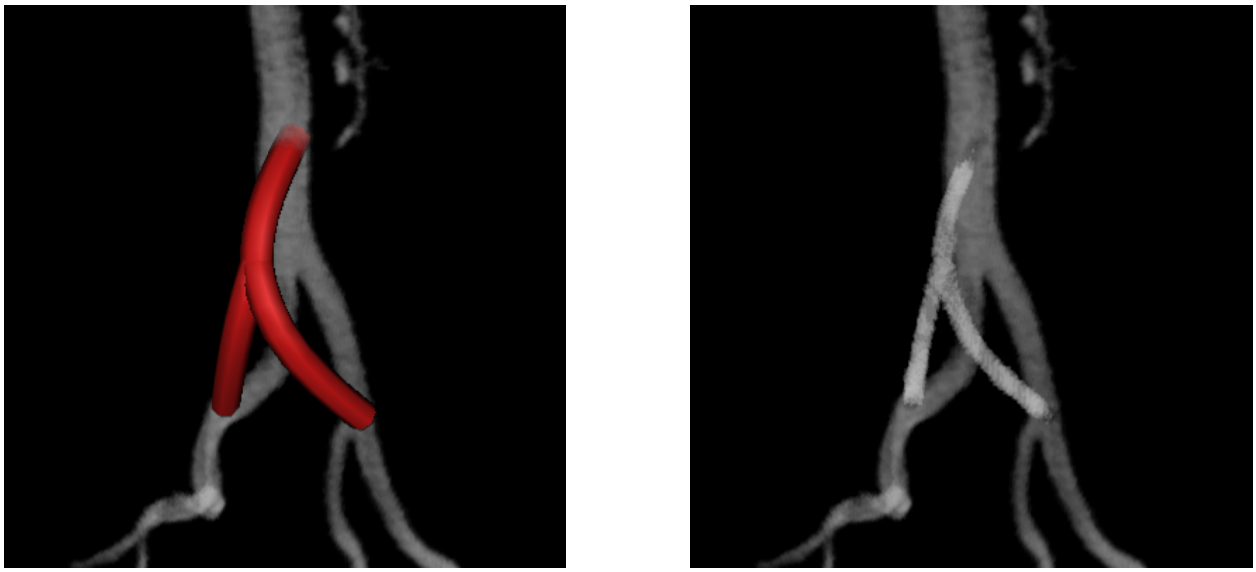


Figure 6.8: Example application of distance sampling to approximate a vascular reconstructive procedure used in abdominal aortic aneurysm repair known as a "aorto-femoral bypass graft with proximal end to side anastomosis" [233]. The picture on the left shows a spline representation of the bypass placed onto the aorta and both iliacs. The picture on the right shows a distance sampling of the bypass, added to the original data set.

Figure 6.8 shows an example of how the distance sampling technique can be used to approximate the placement of a bypass known as an "aorto-femoral bypass graft with proximal end to side anastomosis" [233]. This procedure is applied in patients with aneurysms that involve the aorta and the iliac arteries as well.

Grid generation and editing using stencils

The distance sampling method is well suited to approximate vascular reconstruction procedures that can be described by simple curves, such as shunts or bypasses with a constant diameter. Shapes that are more complex, however, can not be easily created in this manner.
Stencils are patterns that are used to mask (copy or replace) pixels in an image in location where pixels in the stencil are set. In its most simple form a stencil is an image by itself. The previously described distance sampling technique can be used to generate stencils from other shapes that are not images. To change an image, the stencil pattern is transformed to the desired location of the image. Next, each voxel in that image that overlaps a set voxel in the stencil is either replaced or copied to respectively delete or add pixels in the image. Figure 6.9 illustrates this process with a 2D stencil on a 2D image; the process is identical for 3D stencils and images.

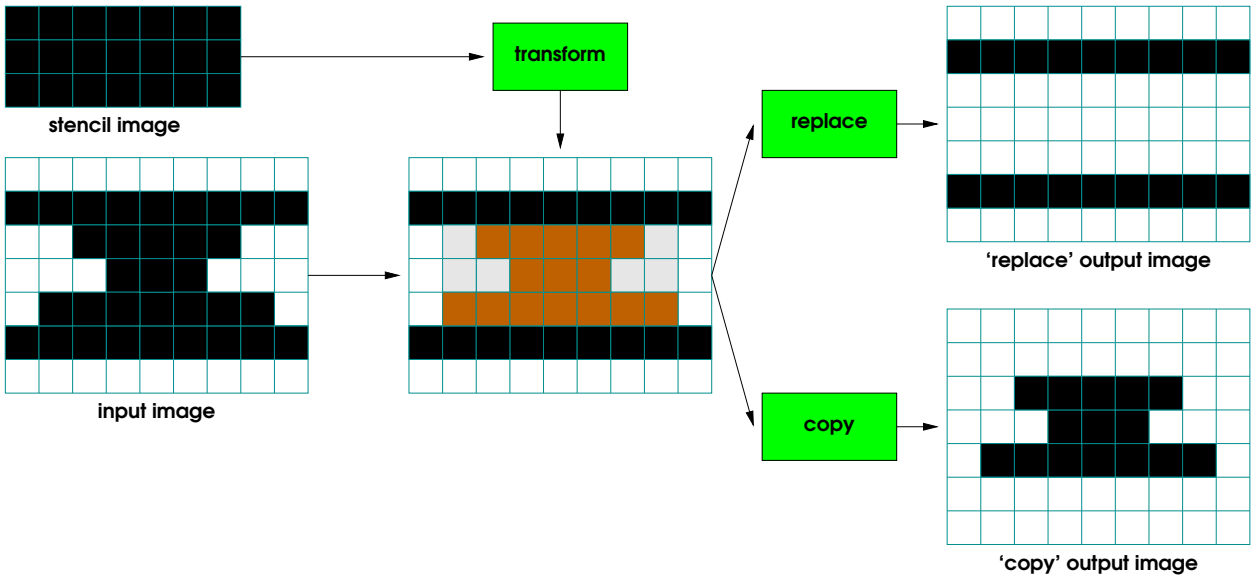


Figure 6.9: The stencil operator used to copy or replace regions of an image.

Grid generation using implicit functions

Implicit functions are functions that, given a 3D coordinate (x,y,z) coordinate, evaluate to a value $w = f_v(x,y,z)$ and a gradient vector $\vec{v} = f_g(x,y,z)$. A carefully structured complex polygon structure, such as those modeled using computer aided design (CAD) or 3D modeling packages, can be used as an implicit function by determining, for each

3D voxel in an output grid; (a) the distance from the voxel to the nearest polygon in the model and (b) the vector to this polygon. This information can be used to discretise a 3D polygonal representation of an object onto a 3D image, as illustrated by the example in Figure 6.10. This Alias|Wavefront model of an airplane was sampled onto a grid of $64 \times 64 \times 64$ voxels using an implicit function operator in Vtk.

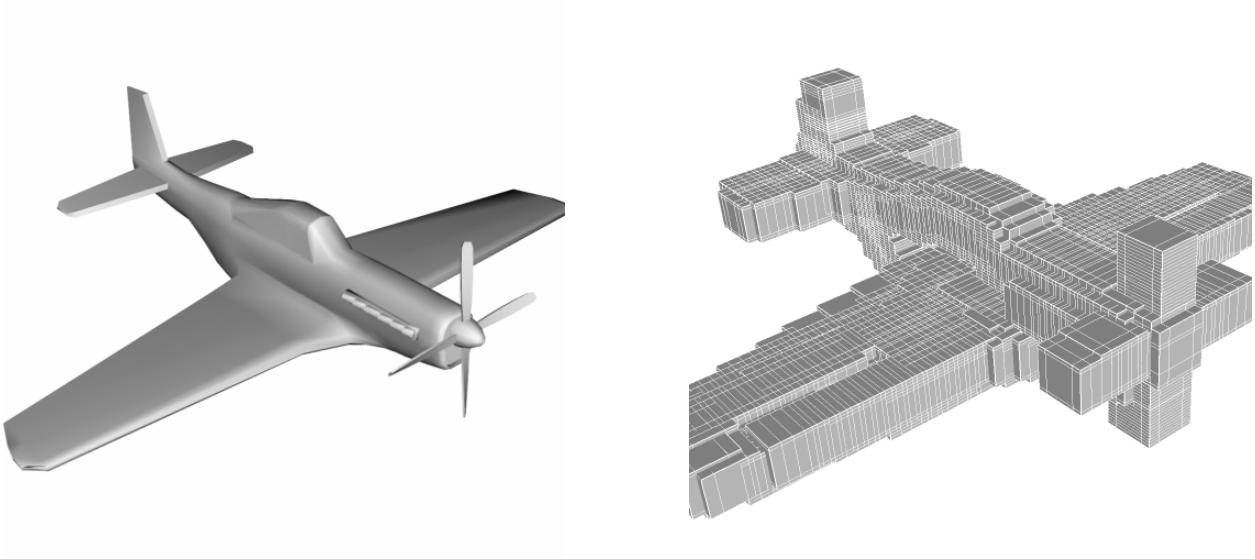


Figure 6.10: A model of an airplane sampled onto a grid of $64 \times 64 \times 64$ *voxels using an implicit function.*

6.2.5 Interactive LBM grid editing in a VE

Surgical procedures in a VE are simulated using a combination of the 3D manipulation techniques offered by SCAVI, described in section 4.2, and the grid generation techniques described in the previous sections. This combination allows a user to interactively add and/or delete areas in the LBM grid corresponding to the procedure that is simulated. Similar grid generation techniques as described above are used to ensure the grids comply to the demands imposed by LBM. Figure 6.11 shows an example of the interactive placement of a simulated bypass, represented by a spline. SCAVI is used to allow the user to manipulate the spline and its control points, represented by spheres, to set the start and end point and the path of the spline. Once the bypass is in place, a distance sampling method is used to sample the bypass and add it onto the existing grid, as previously illustrated in Figure 6.8. After adding the boundary nodes this grid can be converted to a new LBM grid which can be used as the input for a new flow simulation.

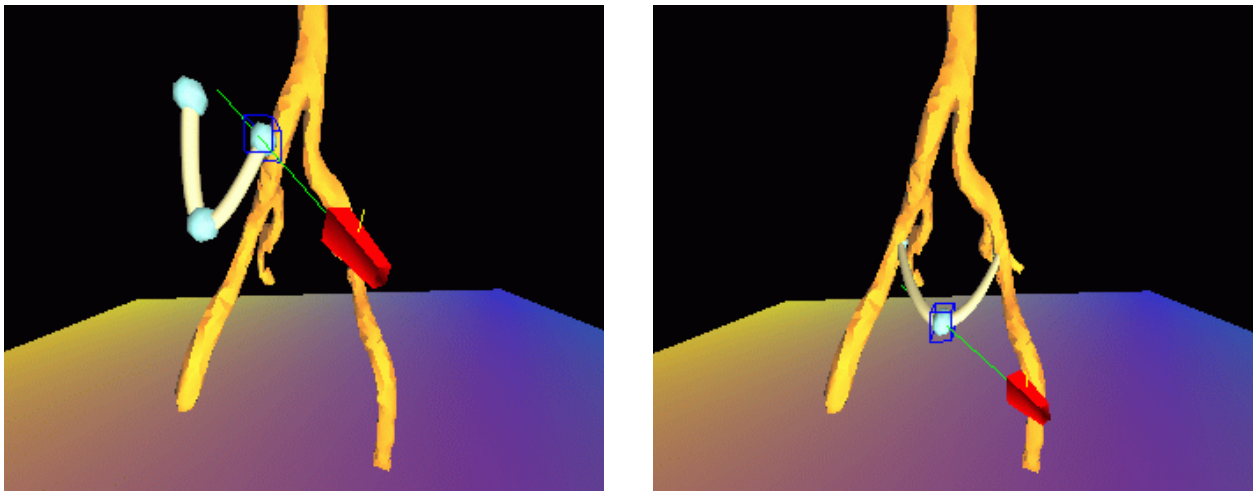


Figure 6.11: Interactive placement of a bypass, represented by a spline. The start and end point of the spline and its path is adjusted using the control points represented by spheres.

6.3 Interactive 3D flow visualization in VEs

The visualization of flow fields is difficult. The challenge in the visualization of flow fields (or vector fields in general) lies in the mapping of vectors to comprehensible visual constructs. Various methods have been designed that do this [56, 57, 185, 246]. The effectiveness of a vector field visualization method depends on the spatial and temporal complexity of the underlying vector field. Although very effective visualization methods have been developed for the representation of 2D vector fields, these methods are not always suitable for 3D fields. The reason for this is often that the 2D visualization methods lack the depth cues that make them suitable for 3D vector field visualization. One way through which these depth cues can be added is through the use of advanced graphics rendering techniques (depth of field, shadows) and VR display and interaction technology (in particular stereoscopic projection and head motion parallax) .

For the purpose of our test-case, we are concerned with the visualization of flow properties that help a surgeon in making a decision on a vascular treatment and deciding what the effect is of a proposed treatment on blood flow [225]. The flow properties that should be represented at minimum are flow direction, flow speed and pressure. We have designed and implemented a number of flow visualization methods for use in interactive VEs [193]. These methods make use of vector field visualization methods in Vtk. The interaction with the flow visualization methods is handled by SCAVI (see section 4.2). The flow visualization methods can be categorized into global and local visualization methods.

6.3.1 Global flow visualization methods

Global visualization methods display the overall behaviour of the flow and are represented by isosurface modeling on scalar properties of the flow (like pressure and flow speed) or by a glyph representation. Glyph visualization of flowfields is a technique where a shape is positioned at each vector, oriented in the direction of the vector and (if desired) coloured by an additional scalar property of the flow at the vector location. The shapes that are most often used in this case have length and direction, such as arrows, so that the shape of the flow becomes apparent. This technique is quite effective for 2D vector fields. For 3D fields, the number of vectors quickly clutters the view on the global flow domain. This can be compensated by subsampling the input data set, but in this case care should be taken of undersampling, in which case important information could be missed, and aliasing effects in the case of regular sampling. Aliasing effects can be reduced by applying a random sampling of the vector field.
With a large number of vectors in the vector field, the number of objects that need to be rendered also increases which may impose a significant load on the rendering engine and, as a consequence, lead to a reduced frame rate. This problem can be solved by the use of glyph objects with as few geometric primitives as possible, such as a simple hook consisting of two lines[‡].

6.3.2 Local flow visualization methods

A common method to visualize flow fields is by tracing the path of a virtual particle through the velocity vectors. Stepwise integration of a particle at position $x(t)$ and velocity $v(x(t))$ at time t through a velocity field is obtained by repeated application of the following procedure:

$$x(t+\delta t) = x(t) + \int_t^{t+\delta t} v(x(t))dt$$

The integration is normally done using a second or fourth order Runge-Kutta method. A path trace is computed starting from an initial location $x(0)$ and repeated application of this procedure with a constant or variable time step. The procedure stops when (1) the virtual particle leaves the velocity field, (2) the speed of the virtual particle drops below a predefined threshold, or (3) a predefined number of integration steps have been performed. After this, the positions computed by this procedure can be used to draw a curve through the velocity field. This method can be repeated for more particles, each at a different starting location. For example; by using a set of equidistant start locations on a line segment (called a “rake”), a similar representation can be obtained as with the smoke released from such an instrument used in real wind tunnel experiments.
Although this visualization method produces acceptable results when applied to 2D velocity fields, it quickly creates occluded views when applied to 3D fields. This can

[‡]Note that lines can not be shaded by most rendering engines, thereby removing a depth-cue which may be important in understanding a 3D vector field.

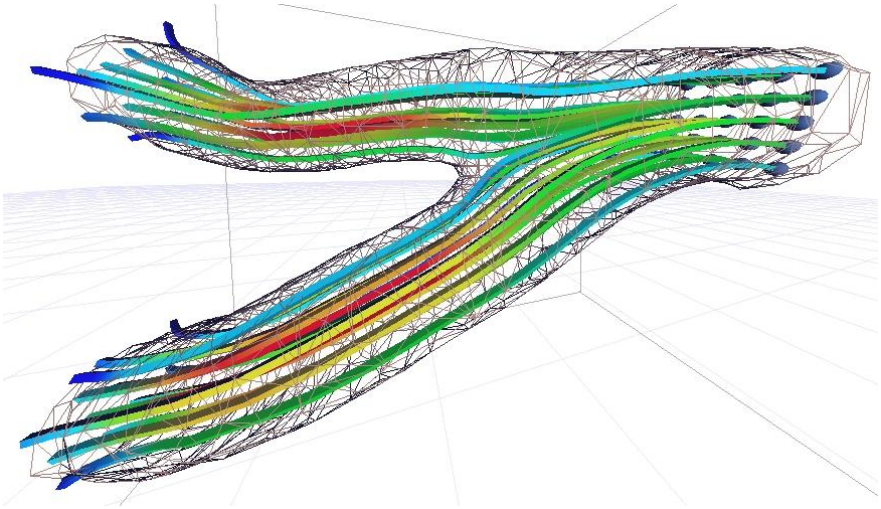


Figure 6.12: Interactive 3D flow visualization in a virtual environment (image created by Hans Ragas). See also colour reproduction on the back cover.

only be partially resolved through depth cues such as stereoscopic projections, motion parallax and shading. Furthermore, it is not easy to tune the integration parameters in such a way that clear results can be obtained.

6.3.3 Results

Figure 6.12 shows an example of an interactive 3D flow visualization method used in a VE. This example shows a streamline representation of a flow field that was simulated through a bifurcated abdominal aorta using the lattice Boltzmann method. The input data set for this simulation was generated from a magnetic resonance angiography (MRA) scan of a healthy patient. This data set was converted into a lattice Boltzmann grid using the methods described in section 6.2.3. The streamlines in this visualization are generated from an interactively moveable plane-shaped source shown at the right of the picture. The colour of the streamlines represents flow speed (from blue for low velocity to red for high velocity). The size, shape and number of particles in the source is interactively controlled by the user from within the environment. The visualization shows that the flow speed is low at the artery walls and higher just after the bifuraction.

6.4 Summary and conclusions

Using the design decisions on the construction of an interactive dynamic exploration environment described in the previous chapters and the additional methods described in this chapter, we have been able to construct an environment that combines fluid flow simulation, visualization and interactive exploration in a virtual environment for the purpose of simulated vascular reconstruction. The environment allows users

to simulate flow through complex shaped geometries and explore the simulation results at run-time. The interactive visualization of results from the simulation in a 3D virtual environment provides insights that would have been difficult to obtain on conventional desktop environments. Furthermore, the interaction capabilities allow the 3D geometry of the geometry to be changed from within the environment to simulate a vascular reconstruction procedure. These changes can then be given back to the fluid flow solver to see the effect of these changes on the flow properties (see also Figure 6.13). Although the system we have developed is by no means ready for use in realistic situations, we have shown that our design choices result in an environment with great potential.

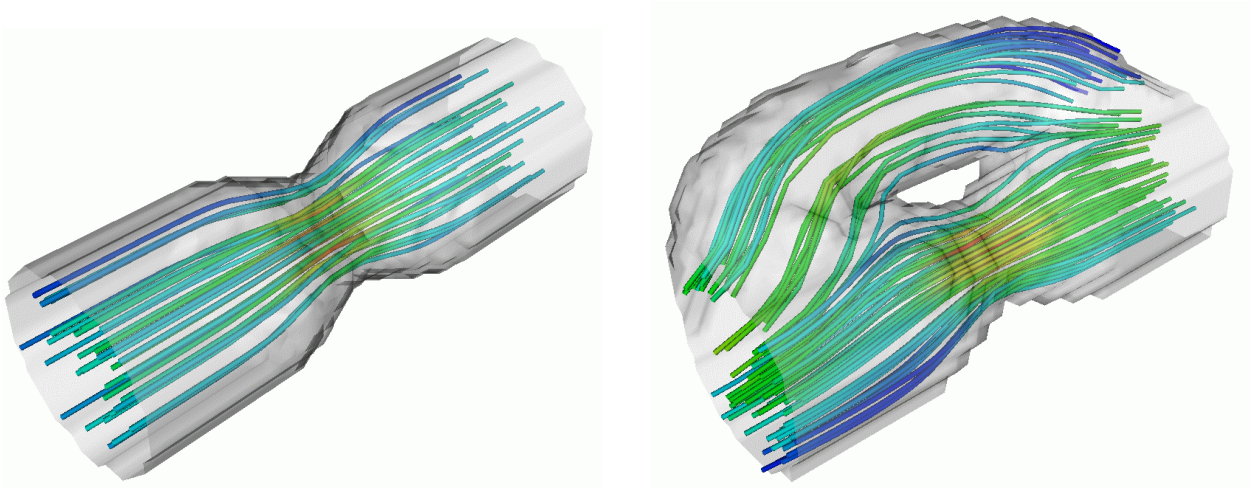


Figure 6.13: The effect of a change in geometry on a flow field (see also colour reproduction on the back cover).

There are however also still several areas that require further attention. Although the 3D flow visualization environment provides an extensive array of possibilities to explore a fluid flow domain, the parameterization of the visualization methods is problematic. An attempt has been made to automatically derive the best parameters for each visualization algorithm, but these will have to be addressed. Also; the current interaction methods to simulate vascular procedures bear little or no resemblance to the actual procedures performed by vascular surgeons. If a system as proposed in this chapter is to be used in realistic situations, the interaction methods will need to be thoroughly reviewed together with prospective users. This process has recently been initiated at the Section Computational Science.

Chapter 7

Summary and concluding remarks

7.1 Summary

Over the last years, computers have become increasingly powerful. As a consequence, researchers are now able to study increasingly large and complex problems. However, this development has led to a situation where the size and complexity of the data spaces that are generated by these problems have increased as well. Often, it turns out to be quite difficult to analyse this data using computer algorithms, either because suitable algorithms do not exist, or because existing algorithms would take too long to find an acceptable result. In these cases, interactive exploration environments may be the only suitable alternative. This thesis makes a distinction between static and dynamic exploration environments. In static environments, the data under study has been generated at an earlier time and will, therefore, not change during exploration. In dynamic environments, the data under study is generated by a computer process that is active *while* the exploration takes place. This would make it possible to not only study the running process, but also alter its course.

In an interactive exploration environment, the data under investigation should be represented to the researcher in an understandable and accurate way. Within the area of scientific visualization, several methods have been developed to accomplish this, but in some cases the complexity of the data can be such that the resulting images are still insufficient to obtain insight in the phenomena under study. In these cases, Virtual Reality (VR) may provide a suitable alternative.

The aim of a a Virtual Environment (VE, the environment that is generated by a VR system) is to immerse the researcher in a representation of his data. To achieve this, the data is represented as a collection of virtual objects in a computer generated artificial world. As in the real world, interaction with these objects enhances the experience of "presence" in the environment, thereby breaking the conventional barrier of a flat computer screen, keyboard and mouse and increasing a researcher's involvement with his data.

Three case studies were built to obtain a better understanding on the available technology, the applications and scientific issues involved in the construction of a usable exploration environment (Chapter 2). These showed that there were still significant

challenges that inhibit the productive use of these environments. The experiences did show that for some applications a VE can help in obtaining a better insight in large and complex data sets.

Until recently, the use of VR required a significant financial investment that was hard to overcome by many. At the end of the Nineties, the performance and price of commodity-of-the-shelf personal computer (PC) hardware was such that it would, in principle, be possible to build a low cost VR system out of easily available components. Chapter 3 describes the design and construction of a PC hardware based VR system and shows that these systems can compete with commercially available solutions.

Essential building blocks were still missing that hampered the construction of a successful interactive exploration environment for scientific research. Chapter 4 describes a number of techniques and interaction methods that address these problems. SCAVI allows the application developer to include existing scientific visualization algorithms into VEs, including methods that allow the researcher to interact directly with these visualizations in an intuitive manner. XiVE bridges a gap by enabling existing 2D desktop applications and graphical user interface toolkits to be used in virtual environments. Automatic speech recognition provides a powerful alternative over graphical interaction methods and, through the use of environment context, the performance of speech recognition can be improved. GEOPROVE allows the researcher to perform measurements on visual entities presented in the virtual environment.

A dynamic environment can benefit from a design where the process under study is separated from the component that represents the data and the component from which the exploration is coordinated. This would allow these different components to execute on dedicated hardware if that could make them perform more efficiently. However, as a consequence, some means of communication must be established between the distributed components. Chapter 5 addresses the issues involved in the design of dynamic environments and describes several architectures that allow these environments to exploit specialized computing resources through a distributed design. Because the exploration environments that are of concern to this research are intended for the exploration of *large* data and problem spaces, the communication performance of these architectures was described in some detail and a number of techniques was described to increase performance.

Based on the design issues addressed in the previous chapters, Chapter 6 describes a test-case environment that has been designed and built for an application from a medical domain: simulated vascular reconstruction. Here, a virtual environment for the representation of 3D angiography scans is connected to a computational fluid solver to simulate blood flow through vascular structures. Interactive scientific visualization techniques allow the researcher to explore the morphology of the structures and the flow behaviour through these structures. Additional interaction techniques allow the researcher to modify the vascular morphology. This environment allows human-in-the-loop simulations through which the researcher can simulate the influence of a proposed treatment on blood flow for a particular patient.

7.2 Concluding remarks

Our description of the construction of conglomerate applications out of distributed executing components in Chapter 5 is currently under review within the Section Computational Science. Several research initiatives are being explored that can help in the development of collaborative interactive dynamic exploration environments for the construction of generic Problem Solving Environments (PSEs). One such initiative is the "Open Grid Services Architecture" (OGSA) that builds on concepts and technologies of the Grid and Web services community in order to define an architecture that provides uniform exposed service semantics [82, 83].
The description of the medical application in Chapter 6 primarily focuses on the software architecture of an environment that can assist a surgeon in his decision process. Although we addressed some of the user interface issues in this design, we have not looked in detail at the usability issues of the particular VR configuration that is most suitable for this application. Clearly, one particular VR configuration will not be suitable for every imaginable application. Current work in our research group includes initiatives to design and build alternative displays, interaction devices and interaction methods that provide a more intuitive interface to the application at hand.

Appendices

Appendix A

The CAVE library

The CAVE Library (CAVELib) is a set of libraries designed as a base for developing virtual reality applications for spatially immersive displays [249]. CAVELib has been in development since the debut of the CAVE in 1992, and is widely used in research and academic areas. CAVELib abstracts from the hardware used in a VR device and therefore allows portable VE applications to be built. The operating systems on which CAVELib is supported is currently IRIX, Linux, Solaris, HPUX and Microsoft Windows. The graphics interfaces supported by CAVELib are OpenGL and Performer. A schematic diagram of a CAVELib application is shown in Figure A.1. CAVELib forks separate processes for each display (i.e. each wall). Communication between processes is done through shared memory which implies that semaphore locking operators are required to enforce mutual exclusion to shared data structures and to synchronize processes. The states of the trackers and controllers are also stored in shared memory. This tracker data is acquired from the hardware through a separate daemon process known as `trackd`. Several library functions are provided by CAVELib to access this data. Each display process is synchronized and automatically renders the correct perspective for each wall. Flexible configuration files make programs written with CAVELib portable to several display and input devices without the need to recompile. The library also provides functions to share information on a virtual environment over a network. This allows users on geographically different locations to collaborate in the same virtual environment.

A.1 Interprocess communication in a CAVELib application

After initialization, CAVELib applications spawn multiple processes; one "master process" that handles user input and performs the necessary calculations that define the behaviour of the virtual environment and one or more "display processes" that do the rendering, one process for each wall. From a software design standpoint the master process describes the process under investigation by filling in (writing) a data structure that is used (read) by the display processes for representation. Conceptually this

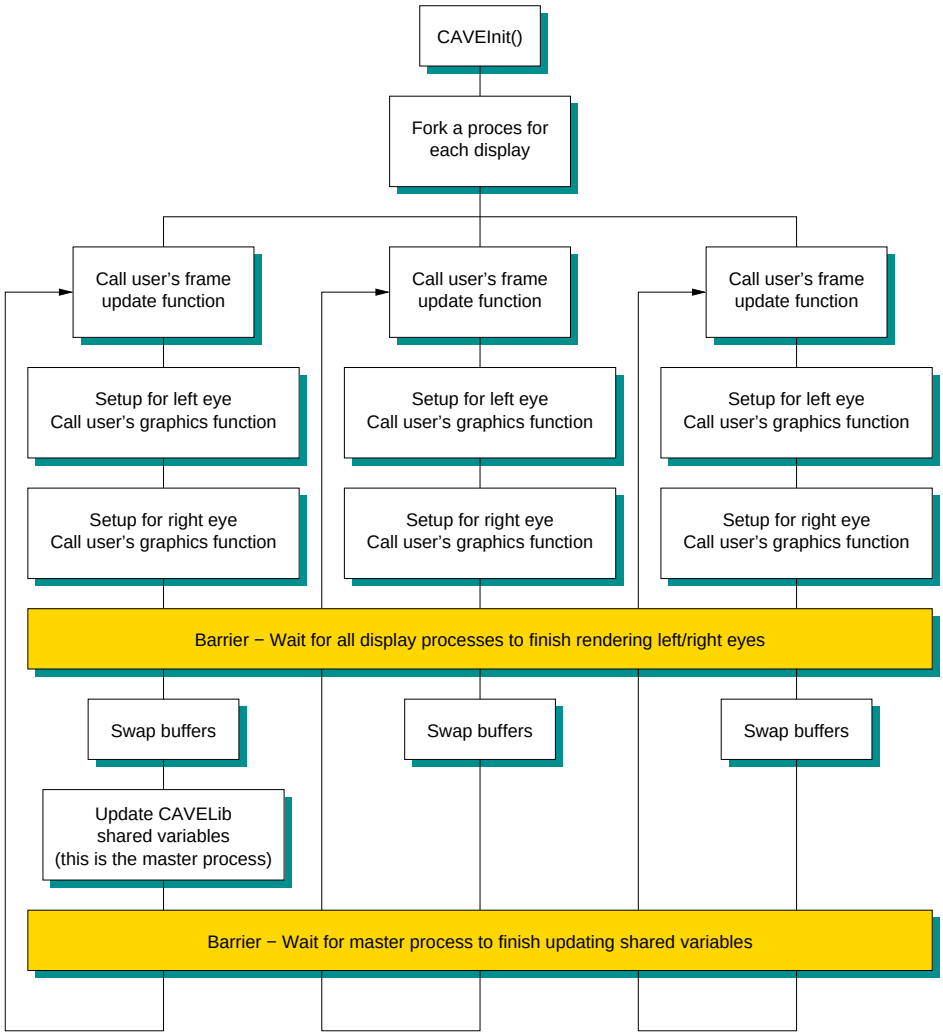


Figure A.1: Flowchart of a CAVELib program [249].

use of a shared data structure ("shared" in the sense that changes made by one process will also be available to other processes) is an easy to understand and sensible program structure to have two processes intercommunicate. Unfortunately, a design choice in CAVELib makes this a little harder than it should be.

The processes created by CAVELib are spawned using the `fork()` system call which means that, at least initially, each process is the same except for its process identifier and parent process identifier. Most UNIX operating systems optimize the creation of new processes by `fork()` by a "copy-on-write" mechanism [232]. This mechanism allows the kernel to create the processes in such a way that they can execute from the same memory areas as long as the processes perform only read accesses to this memory. When a process attempts a write access, the kernel first copies the memory areas before the process is allowed to continue. The implication of this for CAVELib applications is that the "shared" data structure proposed in the previous section will in practice not be shared between the processes at all. Instead, the main process

will have its own private copy from the moment it first writes to it while the display processes are reading from the initial, unchanged version.
The proposed method in CAVELib applications for solving this is through the use of "shared memory". As the name suggests, shared memory allows two or more processes to share a given region of memory [226]. A CAVELib application would, before any processes are spawned, allocate an area of shared memory and then pass a reference to this area to all processes that need access to it. While the main process is writing in the area, semaphores are used to prevent the display processes from accessing the area until the main process is done. The use of shared memory is a perfectly acceptable method for interprocess communications. However, shared memory is a restricted resource in some UNIX kernels. For example; on IRIX systems the maximum amount of shared memory is 4 GB but in most Linux kernels the maximum is set to 32 MB by default. While 4 GB of shared memory is sufficient for most CAVELib applications, the amount of 32 MB is quickly depleted as applications grow more complex.
Another reason why interprocess communication via shared memory is cumbersome is in the overhead that is caused when data needs to be copied from heap (i.e., "normal") memory to shared memory. This can occur when, for example, the main process uses external, third-party libraries to prepare data for rendering that return the output into heap memory (as most libraries do if they use the standard `malloc()` family of functions or the C++ `new` operator). Since the display processes will not be able to access this memory without causing a memory violation, the main process will have to copy this data into shared memory. The overhead caused by this, both in terms of the length of time needed to copy as well as the additional memory needed for the copy, can dramatically influence application performance, especially in dynamic applications where the data that is to be rendered changes frequently.
An easier solution to solve these problems would be if CAVELib spawned *threads* instead of processes [130]. The main difference between processes and threads is that threads share (most of) the resources with its parent process. This allows threads to access memory areas concurrently with its parent process and other threads without the side effects described above. VRCO apparently realized this themselves as since version 3.0 (which was in beta at the beginning of 2002*) CAVELib provides an option to spawn the different processes as threads instead of processes.

*Thanks to Matt Szymanski (VRCO) for allowing me to test a Linux beta version of a multi-threaded CAVELib.

References

[1] 3rdTech webpage. `http://www.3rdtech.com/`.

[2] M.J. Ackerman. Accessing the visible human project. *D-Lib Magazine: The Magazine of the Digital Library Forum*, October 1995.

[3] H. Afsarmanesh, R.G. Belleman, A.S.Z. Belloum, A. Benabdelkader, J.F.J. van den Brand, G.B. Eijkel, A. Frenkel, C. Garita, D.L. Groep, R.M.A. Heeren, Z.W. Hendrikse, L.O. Hertzberger, J.A. Kaandorp, E.C. Kaletas, V. Korkhov, C.T.A.M. de Laat, P.M.A. Sloot, D.Vasunin, A. Visser, and H.H. Yakali. VLAM-G: A grid-based virtual laboratory. *Scientific Programming (Special issue on Grid Computing)*, 10(2):173–181, 2002. ISSN 1058-9244.

[4] Kurt Akeley. The Silicon Graphics 4D/240GTX superworkstation. *IEEE Computer Graphics and Applications*, 9(4):71–83, July 1989.

[5] Nataraj Akkiraju, Herbert Edelsbrunner, Ping Fu, and Jiang Qian. Viewing geometric protein structures from inside a CAVE. *IEEE Computer Graphics and Applications*, pages 58–61, July 1996.

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

[7] I. Angus and H. Sowizral. VRMosaic: Web access from within a virtual environment. *IEEE Computer Graphics and Applications*, pages 6–10, May 1996.

[8] A.M.M. Artoli, B.D. Kandhai, H.G. Hoefsloot, A.G. Hoekstra, and P.M.A. Sloot. Shear stress in lattice boltzmann simulations. In F. Hossfeld and K. Binder, editors, *Europhysics Conference on Computational Physics. Aachen, Germany, 5-8 September 2001. Book of Abstracts.*, volume 8 of *Publication Series of the John von Neumann Institute for Computing*, page A127. John von Neumann Institute for Computing, 2001. abstract.

[9] T. Asano, D.Z. Chen, N. Katoh, and T. Tokuyama. Polynomial-time solutions to image segmentation. In *Proceedings of the 7th Annual SIAM-ACM Conference on Discrete Algorithms*, pages 104–113, January 1996.

[10] David Barberi. The ultimate turing test. On the web: `http://www.ibiblio.org/dbarberi/vr/ultimate-turing/`.

[11] John Perry Barlow. Private life in cyberspace. *Communications of the ACM*, 34(8):23–25, August 1991.

[12] Cagatay Basdogan, Chih-Hao Ho, and Mandayam A. Srinivasan. Simulation of tissue cutting and bleeding for laparoscopic surgery using auxiliary surfaces. In James D. Westwood, Helene M. Hoffman, Richard A. Robb, and Don Stredney, editors, *Medicine Meets Virtual Reality*, pages 38–44, Amsterdam, 1999. IOS Press.

[13] R.G. Belleman, J.A. Kaandorp, and P.M.A. Sloot. A virtual environment for the exploration of diffusion and flow phenomena in complex geometries. *Future Generation Computer Systems*, 14(3-4):209–214, 1998.

[14] R.G. Belleman, B. Stolk, and R. de Vries. Immersive virtual reality on commodity hardware. In R.L. Lagendijk, J.W.J. Heijnsdijk, A.D. Pimentel, and M.H.F. Wilkinson, editors, *Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging, Heijen, the Netherlands*, pages 297–304, Delft, May 30-June 1 2001. Advanced School for Computing and Imaging (ASCI). ISBN 90-803086-6-8.

[15] Carol Bick. Abdominal aortic aneurysm repair. *Nursing Standard*, 15(3):47–52, 2000.

[16] OpenGL Architecture Review Board. *OpenGL Reference Manual (second edition)*. Addison-Wesley, 1996. ISBN 0-201-46140-4.

[17] Maarten Boasson. Control systems software. *IEEE Transactions on Automatic Control*, 38(7):1094–1106, July 1993.

[18] Maarten Boasson. 1,000,000 cooperating processes in a fault tolerant distributed environment, 1997. Sheets from the talk presented at the Distributed Computing (DC'97) workshop, Amsterdam, the Netherlands.

[19] Uli Bockholt, Ulrich Ecke, Wolfgang Müller, and Gerrit Voss. Realtime simulation of tissue deformation for the nasal endoscopy simulator (NES). In James D. Westwood, Helene M. Hoffman, Richard A. Robb, and Don Stredney, editors, *Medicine Meets Virtual Reality*, pages 74–75, Amsterdam, 1999. IOS Press.

[20] John Bohannon. The human genome in 3D at your fingertips. *Science*, 298(5594):737, 25 October 2002.

[21] R.A. Bolt. Put that there: voice and gesture at the graphics interface. *Computer Graphics*, 14(3):262–270, 1980.

[22] Doug A. Bowman. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Symposium on Interactive 3D Graphics*, 1997.

[23] Doug A. Bowman and Larry F. Hodges. User interface constraints for immersive virtual environment applications. In *Proceedings of IEEE VRAIS*, pages 35–38, 1997.

[24] Doug A. Bowman, Donald B. Johnson, and Larry F. Hodges. Testbed evaluation of virtual environment interaction techniques. *Presence: Teleoperators and Virtual Environments*, 10(1):75–95, 2001.

[25] Rachael Brady, John Pixton, George Baxter, Patrick Moran, Clinton S. Potter, Bridget Carragher, and Andrew Belmont. Crumbs: a virtual environment tracking tool for biological imaging. In Murray Loew and Nahum Gurshon, editors, *Proceedings of the IEEE Symposium on Frontiers in Biomedical Visualization*, pages 18–25, Los Alamitos, CA, October 30 1995. IEEE Computer Society Press.

[26] Marco Brassé and Nico Kuijpers. Standardising distributed simulations: The high level architecture. *Xootic Magazine*, 7(1):16–24, July 1999.

[27] K.W. Brodlie, L.A. Carpenter, R.A. Earnshaw, J.R. Gallop, R.J. Hubbold, A.M. Mumford, C.D. Osland, and P. Quarendon. *Scientific Visualization - Techniques and Applications*. Springer Verlag, 1992. ISBN 0-3875-4565-4.

[28] Frederick P. Brooks Jr., Ming Ouh-Young, James J. Batter, and P. Jerome. Project GROPE - haptic displays for scientific visualization. *Computer Graphics*, 24(4):177–185, August 1990.

[29] Steve Bryson. Virtual reality in scientific visualization. *Communications of the ACM*, 39(5):62–71, 1996.

[30] Steve Bryson and Michael J. Gerald-Yamasaki. The distributed virtual windtunnel. In *Supercomputing*, pages 275–284, 1992.

[31] Steve Bryson and Sandy Johan. Time management, simultaneity and time-critical computation in interactive unsteady visualization environments. In *Proceedings of Visualization '96*, page 255. IEEE Computer Science Press, Los Alamitos, CA, 1996.

[32] Steve Bryson and Creon Levit. The virtual windtunnel: An environment for the exploration of three dimensional unsteady flows. Technical Report RNR-92-013, NAS Systems Division, NASA Ames Research Center, 1992.

[33] Grigore Burdea. Virtual reality and robotics in medicine. In *IEEE International Workshop on Robot and Human Communication (RoMAN'96) (Invited Plenary Talk)*, Tsukuba, Japan, November 1996.

[34] H.G. Burkitt, C.R.G. Quick, and D. Gatt. *Essential Surgery*. Churchill Livingstone, 1996. ISBN 0-443-04805-3.

[35] J. Butterworth, A. Davidson, S. Hench, and T.M. Olano. 3DM: A three-dimensional modeler using a head-mounted display. In *ACM Computer Graphics: Proceedings of 1992 Symposium on Interactive 3D Graphics*, pages 135–138, Cambridge, MA, 1992.

[36] Pietro Buttolo, Roberto Oboe, and Blake Hannaford. Architectures for shared haptic virtual environments. *Computers & Graphics*, 21(4):421–429, 1997.

[37] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization*, pages 91–98, New York, NY, October 1994. ACM Press.

[38] AT&T Laboratories Cambridge. VNC - the virtual network computing system. On the web:
`http://www.uk.research.att.com/vnc/`.

[39] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, CA, 1999. ISBN 1-55860-533-9.

[40] C. Carlsson and O. Hagsand. DIVE - a platform for multi-user virtual environments. *Computers and Graphics*, 17(6), 1993.

[41] Jean-Louis Coatrieux, Pascal Haigron, Jean-Louis Dillenseger, and Ivo Stanghellini. From algorithms to applications in medical imaging: present and future. In *State of the Art Reports (INRIA), Eurographics '96*, pages 1–45, Futuroscope Poitiers, France, 26-30 August 1996. ISSN 1017-4656.

[42] CommWeb.com. Automated speech recognition, September 14, 2000. On the web: `http://www.commweb.com/article/COM20000914S0008`.

[43] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics*, volume 26, pages 183–188, 1992.

[44] HOISe-NM Consortium. Dynamite webpage. On the web:
`http://www.hoise.com/dynamite/`.

[45] Ascension Technology Corporation. Ascension Flock of Birds product description. On the web:
`http://www.ascension-tech.com/products/flockofbirds.php`.

[46] Ascension Technology Corporation. Ascension MotionStar Wireless product description. On the web: `http://www.ascension-tech.com/products/motionstarwireless.php`.

[47] IBM Corporation. IBM voice systems: ViaVoice software. On the web: `http://www.ibm.com/viavoice/`.

[48] Immersion Corporation. Immersion Corporation Cyberglove product description. On the web: `http://www.immersion.com/products/3d/interaction/cyberglove.shtml`.

[49] StereoGraphics Corporation. CrystalEyes product information. On the web: `http://www.stereographics.com/products/crystaleyes/body_crystaleyes.html`.

[50] C. Cruz-Neira, T. A. DeFanti, R. Stevens, and P. Bash. Integrating virtual reality and high performance computing and communications for real-time molecular modeling. In *Proceedings of High Performance Computing '95*, April 1995.

[51] C. Cruz-Neira, J. Leigh, C. Barnes, S.M. Cohen, S. Das, R. Engelmann, R. Hudson, M.E. Papka, T. Roy, L. Siegel, C. Vasilakis, T.A. DeFanti, and D.J Sandin. Scientists in wonderland: A report on visualization applications in the cave virtual reality environment. In *Proceedings of IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, pages 59–66, October 1993.

[52] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *SIGGRAPH '93 Computer Graphics Conference*, pages 135–142. ACM SIGGRAPH, August 1993.

[53] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, R.V. Kenyon, and J.C. Hart. The CAVE: Audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):65–72, June 1992.

[54] Marek Czernuszenko, Daniel Sandin, and Thomas DeFanti. Line of sight method for tracker calibration in projection-based VR. In *2nd International Projection Technology Workshop*, May 11-12 1998. `http://www.evl.uic.edu/marek/ipt98/0.html`.

[55] Rudolph P. Darken. Wayfinding in large-scale virtual worlds. In *CHI '95 Conference Proceedings*, pages 45–46, May 7-1, 1995.

[56] Willem C. de Leeuw and Jarke J. van Wijk. A probe for local flow field visualization. In R.D. Bergeron G.M. Nielson, editor, *IEEE Visualization '93*, pages 39–45, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[57] Wim C. de Leeuw and Jarke J. van Wijk. Enhanced spot noise for vector field visualization. In *IEEE Visualization*, pages 233–239, 1995.

[58] J.F. de Ronde. *Mapping in High Performance Computing - A Case Study in Finite Element Simulation*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1998. Promotor: Prof. Dr. P.M.A. Sloot, co-promotor: Prof. Dr. L.O. Hertzberger.

[59] Paul Dechering, Rix Groenboom, Edwin de Jong, and Jan Tijmen Udding. Formalization of a software architecture for embedded systems: a process algebra for splice. In *Proceedings of the 32nd Hawaii International Conference on System Sciences - Volume 3*. IEEE Computer Society, January 1999.

[60] Defense Modeling and Simulation Office (DMSO), Alexandria, VA. *High Level Architecture Run Time Infrastructure Programmer's Guide (1.3 NG version 6)*, 2002. On the web (note: access requires registration and approval by the Software Distribution Center - see `https://sdc.dmso.mil/`): `https://sdc.dmso.mil/contents/rting13v6/ProgrammersGuide.pdf`.

[61] Defense Modeling and Simulation Office (DMSO), Alexandria, VA. *High Level Architecture - Introduction to HLA*, April 1999. CD-ROM Presentation.

[62] Defense Modeling and Simulation Office (DMSO), Alexandria, VA. *High Level Architecture - Time Management*, April 1999. CD-ROM Presentation.

[63] Defense Modeling and Simulation Office (DMSO), Alexandria, VA. *High Level Architecture - Runtime Infrastructure Primer (Corresponding to HLA Interface Specification 1.3)*, December 1998. CD-ROM Presentation.

[64] Defense Modeling and Simulation Office (DMSO), Alexandria, VA. *High Level Architecture - Adapting Your Simulation to HLA*, November 1998. CD-ROM Presentation.

[65] Michael Dinsmore, Noshir Langrana, Grigore Burdea, and Jumoke Ladeji. Virtual reality training for palpation of subsurface tumors. In *IEEE International Symposium on Virtual Reality and Applications (VRAIS'97)*, Albuquerque, New Mexico, March 1997.

[66] Jose Dionisio, Volker Henrich, Udo Jakob, Alexander Rettig, and Rolf Ziegler. The virtual touch: Haptic interfaces in virtual environments. *Computers & Graphics*, 21(4):459–468, 1997.

[67] Penelope F. Grammer (DMSO). Decision by the Defense Modeling and Simulation Office (DMSO) to commercialize the RTI and stop free distribution on September 30, 2002. Prior to September 30, 2002, the implementation of the RTI developed by the Defense Modeling and Simulation Office (DMSO) was freely available for download from the HLA Software Distribution Center. Since

that date, DMSO decided to stop free distribution of HLA "*with the realization that there is an emerging commercial base of RTI development and support that will be better able to meet the future needs of the DoD M&S user community than a single product developed within the DoD*" (e-mail communication via the HLA mailling list, August 2, 2002). The last freely available and supported RTI released by DMSO was version 6 of the RTI 1.3 Next Generation (RTI 1.3 NG V6). Various versions are available from commercial vendors (MÄK Technologies, `www.mak.com`; Mitsubishi Space Software Company, LTD., Japan, `nyumi@kbo.mss.co.jp`; Pitch AB of Sweden and Aegis Technology Group, `dscheiding@aegistg.com`; Science Application International Corporation (SAIC) and Virtual Technology Corporation (VTC), `sturner@virtc.com`).

[68] Kees van den Doel, Paul G. Kry, and Dinesh K. Pai. FoleyAutomatic: Physically-based sound effects for interactive simulation and animation. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 537–544, 2001.

[69] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *ACM Computer Graphics (SIGGRAPH '88)*, 22(4):65–74, August 1988.

[70] Phillip Dykstra. X11 in virtual environments: Combining computer interaction methodologies. In *X Resource issue Nine*. O'Reilly and Associates, 8th Annual X Technical Conference, January 1994.

[71] Niklas Elmqvist. 3Dwm: The three-dimensional workspace manager (next-generation graphical user interfaces on unix platforms). Technical report, Chalmers Medialab, Chalmers University of Technology, Sweden, 2002. On the web:
`http://www.3dwm.org/docs/3dwm-nextgen.pdf`.

[72] T. Todd Elvins. Volume visualization in a collaborative computing environment. *Computers & Graphics*, 20(2):219–222, 1996.

[73] T. Todd Elvins. Virtually lost in virtual worlds – wayfinding without a cognitive map. *ACM Computer Graphics*, 31(3):15–17, 1997.

[74] T. Todd Elvins, David R. Nadeau, and David Kirsh. Worldlets - 3D thumbnails for wayfinding in virtual environments. In *UIST 97*, pages 21–30, 1997.

[75] France ESI Group. Esi group homepage. On the web:
`http://www.esi-group.com/`.

[76] Niessen et al. Nonlinear multicscale representations for image segmentation. *Computer Vision and Image Understanding*, 66:233–245, 1997.

[77] R.F. Farr and P.J. Allisy-Roberts. *Physics for Medical Imaging*. W.B. Saunders Company Ltd., 1998. ISBN 0-7020-1770-1.

[78] J. Feder. *Fractals*. Plenum Press, New York, London, 1988.

[79] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice (2nd Edition)*. Addison Wesley, Reading, Massachusetts, 1996.

[80] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, University of Tennessee, Knoxville, Tennessee, 1994.

[81] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. J. Supercomputer Applications*, 11(2):115–128, 1997.

[82] Ian Foster and Carl Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999. ISBN 1-55860-475-8.

[83] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid (an open grid services architecture for distributed systems integration), June 22 2002. On the web: `http://www.globus.org/research/papers/ogsa.pdf`.

[84] Stratis Gallopoulos, Elias N. Houstis, and John R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1(2):11–23, 1994.

[85] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns (Elements of Reusable Object-Oriented Software)*. Addison-Wesley, Reading, MA, 1994. ISBN 0-201-63361-2.

[86] Desmond Germans, Hans J.W. Spoelder, Luc Renambot, and Henri E. Bal. VIRPI: A high-level toolkit for interactive scientific visualization in virtual reality. In *Proceedings of the Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE)*, Stuttgart, Germany, May 2001.

[87] William K. Gibson. *Neuromancer*. Ace Books, New York, NY, 1984.

[88] Globus. The globus project homepage. On the web: `http://www.globus.org/`.

[89] Henri Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629, June 1971.

[90] Michael A. Guttman, Elias A. Zerhouni, and Elliot R. McVeigh. Analysis of cardiac function from MR images. *IEEE Computer Graphics and Applications*, pages 30–38, January-February 1997.

[91] Douglas P. Haanpaa and Gerald P. Roston. An advanced haptic system for improving man-machine interfaces. *Computers & Graphics*, 21(4):443–449, 1997.

[92] Helmut Haase, Johannes Strassner, and Fan Dai. VR techniques for the investigation of molecule data. *Computers & Graphics*, 20(2):207–217, 1996.

[93] James K. Hahn, Joe Geigel, Jong Won Lee, Larry Gritz, Tapio Takala, and Suneil Mishra. An integrated approach to sound and motion. *Journal of Visualization and Computer Animation*, 6(2):109–123, 1995.

[94] Matthew Hall. vtk2CAVE - Vtk to CAVE translator software. On the web: `http://zeus.ncsa.uiuc.edu/~mahall/`.

[95] D.L. Hannema. Interaction in virtual reality. Master's thesis, University of Amsterdam, Amsterdam, The Netherlands, September 2001.

[96] A.G. Hauptmann. Speech and gestures for graphic image manipulation. In *Proceedings of the ACM CHI'89 Conference on Human Factors in Computing Systems*, pages 241–245, 1989.

[97] A.G. Hauptmann and P. McAvinney. Gesture with speech for graphics manipulation. *International Journal of Man-Machine studies*, 38(2):231–249, February 1993.

[98] David C. Hemmy, Frans W. Zonneveld, Steven Lobregt, and Keizo Fukuta. A decade of clinical three-dimensional imaging: A review. part I. historical development. *Investigative Radiology*, 29(5):489–496, 1994.

[99] I.P. Howard and B.J. Rogers. *Binocular vision and stereopsis*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-508476-4.

[100] R.J. Hubbold. Interactive scientific visualisation - a position paper. In M. Grave, Y. LeLous, and W.T. Hewitt, editors, *Visualization in Scientific Computing (Proceedings of Eurographics Workshop on Scientific Visualisation, Paris, April 1990)*. Springer-Verlag, April 1994. ISBN 3-540-56147-1.

[101] Roger J. Hubbold, Xiao Dongbo, and Simon Gibson. MAVERIK - the manchester virtual environment interface kernal. In M. Göbel, J. David, P. Slavik, and J. J. van Wijk, editors, *Virtual Environments and Scientific Visualization '96*, pages 11–20. Springer-Verlag Wien, 1996.

[102] Randy Hudson. VRen: a C++ class for VR volume rendering. *Final project report for EECS590*, April 30, 1990.

[103] IBM Corporation, Armonk, NY. *Data Explorer Reference Manual*, 1991.

[104] IBM ViaVoice Software Developer's Kit, IBM Corporation, USA. *SMAPI Reference*, 2000.

[105] IBM ViaVoice Software Developer's Kit, IBM Corporation, USA. *SMAPI User's Guide*, 2000.

[106] Kitware Inc. Visualization toolkit homepage. On the web: `http://www.vtk.org/`.

[107] International Business Machines (IBM) Corp. *IBM ViaVoice Software Development's Kit*. `http://www.ibm.com/software/speech/dev/`.

[108] Jerry Isdale. What is Virtual Reality? On the web: `http://www.isdale.com/jerry/VR/WhatIsVR.html`.

[109] H. Iwata. Volume haptization. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, San Jose, California, October 1993.

[110] Linda Jacobson. *Cyberarts: Exploring Art & Technology*. Backbeat Books, August 1992. ISBN 0879302534.

[111] J.A. Kaandorp. Modelling growth forms of the sponge *haliclona oculata* (Porifera; Demospongiae) using fractal techniques. *Marine Biology*, 110:203–215, 1991.

[112] J.A. Kaandorp. *Modelling growth forms of biological objects using fractals*. PhD thesis, University of Amsterdam, 1992. Promotor: Prof. Dr. Ir. F.C.A. Groen, 2nd promotor: Dr. J.H. Stock, co-promotor: Dr. E.H. Dooijes, 2nd co-promotor: Prof. Dr. H.A. Lauwerier.

[113] J.A. Kaandorp. Fractal modelling of growth forms of the sponge *haliclona oculata*. In J. Demongeot and V. Capasso, editors, *Mathematics applied to biology and medicine*, pages 104–106, Winnipeg, Canada, 1993. Wuerz Publishing Ltd. Proc. first European Conference of Mathematics applied to Biology and Medicine, Grenoble, France.

[114] J.A. Kaandorp. Simulating radiate accretive growth using iterative geometric constructions. In E. Louis, L.M. Sander, P. Meakin, and J.M. Garcia-Ruiz, editors, *Growth patterns in physical sciences and biology, NATO ASI Series B: Physics Vol. 304*, pages 331–340, New York, 1993. Plenum Press.

[115] J.A. Kaandorp. Diffusion limited radiate accretive growth. In R. Gruber and M. Tomassini, editors, *Proceedings of the 6th joint EPS-APS International Conference on Physics Computing*, pages 281–284, Lugano, Switserland, August 1994. European Physical Society, European Physical Society, Geneva, Switserland.

[116] J.A. Kaandorp. A formal description of radiate accretive growth. *Journal of Theoretical Biology*, 166:149–161, 1994.

[117] J.A. Kaandorp. *Fractal modelling: growth and form in biology*. Springer-Verlag, Berlin, New York, 1994.

[118] J.A. Kaandorp. Analysis and synthesis of radiate accretive growth in three dimensions. *J. Theor. Biol.*, 175:39–55, 1995.

[119] J.A. Kaandorp. Morphological analysis of growth forms of branching marine sessile organisms along environmental gradients. *Mar. Biol.*, 134:295–306, 1999.

[120] J.A. Kaandorp and M.J. de Kluijver. Verification of fractal growth models of the sponge *haliclona oculata* (Porifera; class Demospongiae) with transplantation experiments. *Marine Biology*, 113:133–143, 1992.

[121] J.A. Kaandorp, C. Lowe, D. Frenkel, and P.M.A. Sloot. The effect of nutrient diffusion and flow on coral morphology. *Phys. Rev. Lett.*, 77(11):2328–2331, 1996.

[122] D. Kandhai. *Large Scale Lattice-Boltzmann Simulations (Computational Methods and Applications)*. PhD thesis, Universiteit van Amsterdam, Amsterdam, the Netherlands, 1999.

[123] D. Kandhai, A. Koponen, A.G. Hoekstra, M. Kataja, J. Timonen, and P.M.A. Sloot. Lattice Boltzmann hydrodynamics on parallel systems. *Computer Physics Communications*, 1998.

[124] D. Kandhai, D. Vidal, A. Hoekstra, H. Hoefsloot, P. Iedema, and P.M.A. Sloot. Lattice-Boltzmann and finite element simulations of fluid flow in a SMRX mixer. *Int. J. Numer. Meth. Fluids*, 31:1019–1033, 1999.

[125] Paul E. Keller, Richard T. Kouzes, Lars J. Kangas, and Sherif Hashem. Transmission of olfactory information for telemedicine. In *Interactive Technology and the New Paradigm for Healthcare*, pages 168–172. IOS Press, Washington, DC, 1995.

[126] Brian W. Kernighan and Dennis M. Ritchie. *The C programming language (2nd edition)*. Prentice Hall, London, UK, 1988. ISBN 0-13-110362-8.

[127] G. Drew Kessler, Doug A. Bowman, and Larry F. Hodges. The simple virtual environment library: An extensible framework for building VE applications. *Presence: Teleoperators and Virtual Environments*, 9(2):187–208, 2000.

[128] Volodymyr Kindratenko. A survey of electromagnetic position tracker calibration techniques. *Virtual Reality: Research, Development and Applications*, 5(3):169–182, 2000.

[129] Volodymyr Kindratenko. A comparison of the accuracy of an electromagnetic and a hybrid ultrasound-inertia position tracking system. *Presence: Teleoperators and Virtual Environments*, 10(6):657–663, 2001.

[130] Steve Kleiman, Devang Shah, and Bart Smaalders. *Programming with Threads*. Prentice Hall, Mountain View, CA, 1996.

[131] Eugenia M. Kolasinski. Simulator sickness in virtual environments. Technical report, U.S. Army Research Insitute, 1995.

[132] David R. Koller, Mark R. Mine, and Scott E. Hudson. Head-tracked orbital viewing: An interaction technique for immersive virtual environments. In *UIST '96: The Ninth Annual Symposium on User Interface Software and Technology*, Seattle, WA, 1996.

[133] Anton H.J. Koning. Applications of volume rendering in the cave. In B. Engquist et al., editors, *Simulation and Visualization on the Grid*, pages 112–121, Heidelberg, December 1999. Springer Verlag.

[134] A. Koponen, D. Kandhai, E. Hellen, M. Alava, A. Hoekstra, M. Kataja, K. Niskanen, P. Sloot, and J. Timonen. Permeability of three-dimensional random fiber webs. *Physical Review Letters*, 80(4):716–719, January 26 1998.

[135] Michal Koutek and Frits H. Post. Dynamics in interaction on the responsive workbench. In *Proceedings of the sixth Eurographics Workshop on Virtual Environments*, pages 43–54, Amsterdam, June 1-2, 2000.

[136] Glenn E. Krasner and Stephen T. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[137] Myron W. Krueger. *Artificial Reality II*. Addison-Wesley, Reading, MA, 1991. ISBN 0-201-52260-8.

[138] Myron W. Krueger. Olfactory stimuli in virtual reality for medical applications. In *Interactive Technology and the New Paradigm for Healthcare*, pages 180–181. IOS Press, Washington, DC, 1995.

[139] David N. Ku. Blood flow in arteries. *Annu. Rev. Fluid Mech.*, 29:399–434, 1997.

[140] Cameron Laird and Kathryn Soraiz. Xvfb - a conversation every system administrator should hear. Unix Insider, On the web: `http://www.itworld.com/AppDev/1461/UIR000330xvfb/`.

[141] Jaron Lanier. Music from inside virtual reality: The sound of one hand. *Whole Earth Review*, pages 30–34, June 1993.

[142] J. Laviola. MSVT: A virtual reality-based multimodal scientific visualization tool. In *Proceedings of the Second IASTED International Conference on Computer Graphics and Imaging*, pages 221–225, 1999.

[143] Timothy Leary. Virtual reality: How to operate and teleport your brain. Videotape (95 min).

[144] Jonathan Leech, Jan F. Prins, and Jan Hermans. SMD: Visual steering of molecular dynamics for protein design. *IEEE Computational Science and Engineering*, pages 38–45, Winter 1996.

[145] J. Leigh, A. Johnson, and T. DeFanti. Cavern: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality: Research, Development and Applications*, 2(2):217–237, 1997.

[146] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(3):29–37, May 1988.

[147] J.C.R. Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1:4–11, March 1960.

[148] Y.S. Lim, T.I. Cho, and K.H. Park. Range image segmentation based on 2D quadratic function approximation. *Pattern Recognition Letters*, 11:699–708, 1990.

[149] Robert W. Lindeman, John L. Sibert, and James K. Hahn. Towards usable VR: An empirical study of user interfaces for immersive virtual environments. In *CHI '99 Conference Proceedings*, pages 64–71, May 15-20, 1999.

[150] Carl Eugene Loeffler and Tim Andersen. *The Virtual Reality Casebook*. Van Nostrand Reinhold, New York, NY, 1994. ISBN 0-442-01776-6.

[151] W.E. Lorensen and H.E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169, 1987.

[152] Jean loup Gailly and Mark Adler. zlib (homepage). On the web: `http://www.gzip.org/zlib/`.

[153] Greg R. Luecke, Young-Ho Chai, and James C. Edwards. Force interactions in the synthetic environment using the ISU force reflecting exoskeleton. *Computers & Graphics*, 21(4):431–442, 1997.

[154] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz. NPSNET: A network software architecture for large-scale virtual environment. *Presence*, 3(4):265–287, 1994.

[155] B.B. Mandelbrot. *The fractal geometry of nature*. Freeman, San Francisco, 1983.

[156] Michael E. McCauley and Thomas J. Sharkey. Cybersickness: Perception of self-motion in virtual environments. *Presence: Teleoperators and Virtual Environments*, 1(3):311–318, 1992.

[157] Bruce H. McCormick, Thomas A. DeFanti, and M.D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), 1987.

[158] Hilary McLellan. Virtual realities. In David H. Jonassen, editor, *Handbook of research for educational communications and technology*, pages 457–487. Simon & Schuster Macmillan, New York, NY, 1996. ISBN 0-02-864663-0. Online at the Association for Educational Communications and Technology; `http://www.aect.org/Intranet/Publications/edtech/pdf/15.pdf`.

[159] Chalmers Medialab. 3Dwm homepage. On the web: `http://www.3dwm.org/`.

[160] K. Meyer, H. Applewhite, and F. Biocca. A survey of position trackers. *Presence: Teleoperators and Virtual Environments*, 1(2):173–200, 1992.

[161] Mark R. Mine. ISAAC: A virtual environment tool for the interactive construction of virtual worlds. Technical report, University of North Carolina, 1995. Computer Science Technical Report TR95-020.

[162] Mark R. Mine. Virtual environment interaction techniques. Technical report, University of North Carolina, 1995. Technical Report TR95-018.

[163] Mark R Mine. Working in a virtual world: Interaction techniques used in the chapel hill immersive modeling program. Technical Report TR96-029, Department of Computer Science, University of North Carolina, 21, 1996.

[164] Mark R. Mine. Moving cows in space: Exploiting proprioception as a framework for virtual environment interactions. Technical report, University of North Carolina, 1997. Computer Science Technical Report TR97-003.

[165] Defense Modeling and Simulation Office (DMSO). High Level Architecture (HLA) homepage. On the web: `https://www.dmso.mil/public/transition/hla/rti/`.

[166] Judi Moline. Virtual environments for health care – a white paper for the advanced technology program (atp). Technical report, National Institute of Standards and Technology, October 1995. On the web: `http://ovrt.nist.gov/projects/health/health.html`.

[167] Jurriaan D. Mulder. Remote object translation methods for immersive virtual environments. In *Proceedings of Eurographics Virtual Environments '98*. Springer, 1998.

[168] Jurriaan D. Mulder and Robert van Liere. The personal space station: Bringing interaction within reach. In S. Richer, P. Richard, and B. Taravel, editors, *Proceedings of the Virtual Reality International Conference, VRIC 2002*, pages 73–81, 2002.

[169] Atul Nayak, Jason Leigh, and Shalini Venkataram. CAVERNsoft G2, A toolkit for high performance tele-immersive collaboration, 2002. On the web: `http://www.openchannelsoftware.org/projects/CAVERNsoft_G2/`.

[170] Gregory M. Nielson, Hans Hagen, and Heinrich Müller. *Scientific Visualization (Overviews, Methodologies and Techniques)*. IEEE Computer Society Press, 1997. ISBN 0-8186-7777-5.

[171] Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, New York, NY, 1988. ISBN 0-465-06709-3. Also published as *The Design of Everyday Things*, Currency/Doubleday, New York, NY, 1990. ISBN 0-385-26774-6.

[172] The Numerical Algorithms Group Ltd., Oxford, UK. *Iris Explorer User's Guide*, 1998.

[173] nVidia Corporation. nVidia Linux OpenGL drivers, 2002. On the web: `http://www.nvidia.com/view.asp?IO=linux_display_archive`.

[174] National Library of Medicine. Visible human project homepage, 2002. On the web: `http://www.nlm.nih.gov/research/visible/visible_human.html`.

[175] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, 1994. ISBN 0-201-63337-X.

[176] Benno J. Overeinder and Peter M. A. Sloot. Extensions to time warp parallel simulation for spatially decomposed applications. In David Al-Dabass and Russell Cheng, editors, *Proceedings of the Fourth United Kingdom Simulation Society Conference (UKSim 99)*, pages 67–73, Cambridge, UK, April 1999.

[177] Sharon L. Oviatt. Mutual disambiguation of recognition errors in a multimodal architecture. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'99)*, pages 576–583, Pittsburgh, PA, 1999. ACM Press.

[178] Steven G. Parker and Christopher R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Supercomputing '95*, 1995.

[179] Randy Pausch, Tommy Burnette, and Michael E. Weiblen. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. In *UIST '96: The Ninth Annual Symposium on User Interface Software and Technology*, page 518, 1995.

[180] Helen Pearson. 3D liver aids surgery (virtual organ image beamed into OR), March 11, 2002. Nature News Service / Macmillan Magazines. On the web: `http://www.nature.com/nsu/020304/020304-12.html`.

[181] Standard Performance Evaluation Corporation. SPEC CPU benchmark. On the web: `http://www.spec.org/`.

[182] Philips. *Basic Principles of MR Imaging (third edition)*. Philips Medical Systems, 2001.

[183] Bui-Tuong Phong. Illumination for computer generated images. *Communications of the ACM*, 18(6):311–317, June 1975.

[184] Technology Playgroup. Technology Playgroup Unwinder product description. On the web:
`http://this.is/tpg/products/unwinder/`.

[185] Frits H. Post and Jarke J. van Wijk. Visual representation of vector fields: recent developments and research directions. In L. Rosenblum, R.A. Earnshaw, J. Encarnaçao, H. Hagen, A. Kaufman, S. Klimenko, G. Nielson, F. Post, and D. Thalmann, editors, *Scientific Visualization: Advances and Challenges*, pages 367–390. Academic Press, London, 1994. ISBN 0-12-227742-2.

[186] Timothy Poston and Luis Serra. Dextrous virtual work. *Communications of the ACM*, 39(5):37–45, May 1996.

[187] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique: Non-linear mapping for direct manipulation in VR. In *UIST '96: The Ninth Annual Symposium on User Interface Software and Technology*, Seattle, WA, 1996.

[188] Ivan Poupyrev, Suzanne Weghorst, Mark Billighurst, and Tadao Ichikawa. A framework and testbed for studying manipulation techniques for immersive VR. In *Proceedings of ACM Conference VRST*, pages 21–28, 1997.

[189] Silicon Graphics Inc. Software Products. OpenGL homepage. On the web:
`http://www.sgi.com/software/opengl/`.

[190] Silicon Graphics Inc. Software Products. Performer homepage. On the web:
`http://www.sgi.com/software/performer/`.

[191] Silicon Graphics Inc. Software Products. Volumizer homepage. On the web:
`http://www.sgi.com/software/volumizer/`.

[192] Frederick H. Raab, Ernest Blood, Terry O. Steiner, and Herbert R. Jones. Magnetic position and orientation tracking system. *IEEE Transactions on Aerospace and Electronic Systems*, 15(5):709–718, 1979.

[193] J.M. Ragas. Interactive visualization of flowfields in an immersive virtual environment. Master's thesis, University of Amsterdam, Amsterdam, The Netherlands, August 2002.

[194] Luc Renambot, Henri E. Bal, Desmond Germans, and Hans J.W. Spoelder. High-level steering: Measuring in virtual reality environments, August 2000.

[195] Howard Rheingold. *Virtual Reality*. Mandarin paperbacks, London, 1992. ISBN 0-7493-0889-3.

[196] B. Rinkevich and Y. Loya. Coral isomone: a proposed chemical signal controlling interclonal growth patterns in a branching coral. *Bull. Mar. Sci.*, 36:319–324, 1985.

[197] Richard A. Robb. Visualization & Virtual Reality in biomedical applications. Handout for the *Visualization and Virtual Reality* workshop at the *Computer Assisted Radiology and Surgery 2001* (CARS 2001) conference.

[198] W. Robinett and R. Holloway. Implementation of flying, scaling and grabbing in virtual worlds. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 197–208, 1992.

[199] P. Rogalla. Virtual endoscopy: an application snapshot. *Medica Mundi*, 43(1):17–23, March 1999.

[200] Jannick P. Rolland, William Gibson, and Dan Ariely. Towards quantifying depth and size perception in virtual environments. *Presence: Teleoperators and Virtual Environments*, 4(1):24–49, 1995.

[201] Michael Rorke, Shaun Bangay, and Peter Wentworth. Virtual reality interaction techniques. In *Proceedings of the 1st Annual South African Telecommunication, Networks and Application Conference (SATNAC)*, pages 526–532, Cape Town, South Africa, September 1998.

[202] T.M. Roy and T.A. DeFanti. Interactive visualization in a high performance computing virtual environment. In *Proceedings of the 1995 Simulation Multiconference*, pages 471–476. The Society for Computer Simulation, 1995.

[203] Luc Sala and John P. Barlow. *Virtual Reality (de Metafysische Kermisattractie)*. SALA Communications GmbH, Dusseldorf, 1990. ISBN 90-73107-02-4.

[204] Mark S. Sanders and Ernest J. McCormick. *Human Factors in Engineering and Design (7th ed.)*. McGraw-Hill, 1993. ISBN 0-07-054901-X.

[205] SARA Computing and Networking Services, Amsterdam, the Netherlands. *SARA Virtual Reality Facilities; the CAVE*. On the web: `http://www.sara.nl/products/products_08a_01_eng.html`.

[206] J.A. Schaap, P.J.H. de Koning, R.J. van der Geest, and J.H.C. Reiber. 3D quantification and visualization of MRA. In H.U. Lemke, M.W. Vannier, K. Inamura, A.G. Farman, and K.Doi, editors, *Computer Assisted Radiology and Surgery (Excerpta Medica, International Congress Series 1230)*, pages 928–933, Berlin, Germany, June 2001. Elsevier Science B.V.

[207] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The design of the TAO real-time object request broker. *Computer Communications*, 21(4):294–324, 10 April 1998.

[208] Ben Schneiderman. The limits of speech recognition. *Communications of the ACM*, 43(9):63–65, September 2000.

[209] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit, an object-oriented approach to 3D graphics (2nd edition)*. Prentice Hall, Upper Saddle River, NJ, 1997. ISBN 0-13-954694-4.

[210] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, 1992.

[211] K.P. Sebens, J. Witting, and B. Helmuth. Effects of water flow and branch spacing on particle capture by the reef coral *madracis mirabilis* (Duchassaing and Michelotti). *J. Exp. Mar. Biol. Ecol.*, 211:1–28, 1997.

[212] Sense8. Worldtoolkit product webpage. On the web: `http://www.sense8.com/products/wtk.html`.

[213] L. Serra, T. Poston, N. Hern, C. B. Choon, and J. A. Waterworth. Interaction techniques for a virtual workspace. In *Proceedings of VRST'95*, pages 79–80, 1995.

[214] R. Sethi. *Programming Languages: Concepts and Constructs*. Addison-Wesley, Upper Saddle River, NJ, 1989.

[215] R. Sharma, M. Zeller, V.I. Pavlovic, T.S. Huang, Z. Lo, S. Cuh, Y. Zhao, J.C. Philips, and K. Schulten. Speech/gesture interface to a visual-computing environment. *IEEE Computer Graphics and Applications*, pages 29–37, March-April 2000.

[216] Bill Sherman. FreeVR homepage. On the web: `http://www.freevr.org/`.

[217] Silicon Graphics, Inc. Onyx2 Reality, Onyx2 InfiniteReality and Onyx2 InfiniteReality2. Technical report, Silicon Graphics, Inc., Mountain View, CA, August 1998.

[218] Silicon Graphics, Inc., Mountain View, CA. *OpenGL Performer Getting Started Guide*, 2002.

[219] Silicon Graphics, Inc., Mountain View, CA. *OpenGL Performer Programmer's Guide*, 2002.

[220] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments (design and implementation)*. Addison-Wesley, 1999. ISBN 0-201-32557-8.

[221] P.M.A. Sloot, A. Schoneveld, J.F. de Ronde, and J.A. Kaandorp. Large-scale simulations of complex systems, part I: Conceptual framework. Technical report, Santa Fe Institute, 1997. 97-07-070.

[222] Robert Spence. *Information Visualization*. ACM Press, Oxford, UK, 2001. ISBN 0-201-59626-1.

[223] V. Spitzer, M.J. Ackerman, A.L. Scherzinger, and D. Whitlock. The visible human male: a technical report. *J Am Med Inform Assoc*, 3(2):118–130, 1996.

[224] Mandayam A. Srinivasan and Cagatay Basdogan. Haptics in virtual environments: Taxonomy, research status, and challenges. *Computers & Graphics*, 21(4):393–404, 1997.

[225] David A. Steinman. Simulated pathline visualization of computed periodic blood flow patterns. *Journal of Biomechanics*, 33:623–628, 2000.

[226] W. Richard Stevens. *Advanced Programming in the UNIX Environment*. Addison Wesley, Reading, Massachusetts, 1992.

[227] Bjarne Stroustrup. *The C++ programming language (3rd edition)*. Addison Wesley, Upper Saddle River, NJ, 1997. ISBN 0-201-88954-4.

[228] Bernhard Suhm, Brad Meyers, and Alex Waibel. Multimodal error correction for speech user interfaces. *ACM transactions on Computer-Human Interaction*, 8(1):60–98, March 2001.

[229] V.S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[230] I.E. Sutherland. The ultimate display. In *Proceedings of the IFIP Congress, 2*, pages 506–508, 1965.

[231] I.E. Sutherland. A head-mounted three-dimensional display. In *Proceedings of the AFIPS Fall Joint Computer Conference*, volume 33, pages 757–764, 1968.

[232] A. Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, Upper Saddle River, NJ, 1997.

[233] Charles A. Taylor. Simulation-based medical planning systems, 2001. On the web:
`http://www-med.stanford.edu/school/vascular/RESEARCH/Simulation.htm`.

[234] Charles A. Taylor, Thomas J.R. Hughes, and Christopher K. Zarins. Finite element modeling of three-dimensional pulsatile flow in the abdominal aorta: Relevance to atherosclerosis. *Annals of Biomedical Engineering*, 26:975–987, 1998.

[235] D. Tennenhouse. Proactive computing. *Communications of The ACM*, 43(5):43–50, May 2000.

[236] TGS. Open inventor product webpage. On the web:
`http://www.tgs.com/pro_div/oiv_main.htm`.

[237] Henrik Tramberend. Avocado: A distributed virtual reality framework. In *IEEE Virtual Reality*, page 14. IEEE Press, March 1999.

[238] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1999.

[239] Markku Turunen. Error handling in speech user interfaces in the context of virtual worlds. In *Proceedings of ACHCI'98*, pages 68–75, Tampere, Finland, 1998.

[240] Heang K. Tuy and Lee Tan Tuy. Direct 2D display of 3D objects. *IEEE Computer Graphics & Applications*, 4(10):29–33, October 1984.

[241] Iowa State University. VR juggler homepage. On the web: `http://www.vrjuggler.org/`.

[242] C. Upson, T. Faulhaber Jr., and D. Kamins et al. The Application Visualization System: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[243] G.D. van Albada, J. Clinckemaillie, A.H.L. Emmen, J. Gehring, O. Heinz, F. van der Linden, B.J. Overeinder, A. Reinefeld, and P.M.A. Sloot. Dynamite - blasting obstacles to parallel cluster computing. In P.M.A. Sloot, M. Bubak, A.G. Hoekstra, and L.O. Hertzberger, editors, *High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands*, number 1593 in Lecture Notes in Computer Science, pages 300–310, Berlin, April 1999. Springer-Verlag.

[244] R. van Liere and J.D. Mulder. PVR: An architecture for portable VR applications. In *Proceedings EG Workshop on Virtual Environments*, pages 125–135. Springer-Verlag, 1999.

[245] Robert van Liere, Jurriaan D. Mulder, and Jarke J. van Wijk. Computational steering. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking*, pages 696–702. Springer-Verlag, April 1996.

[246] Jarke J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics (special issue, Proceedings ACM SIGGRAPH 2002)*, 21(3):745–754, 2002.

[247] John Vince. *Essential Virtual Reality Fast*. Springer-Verlag, London, 1998. ISBN 1-85233-012-0.

[248] Steve Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[249] Virtual Reality Consulting (VRCO) Inc., Chicago, IL. *CAVE User's Guide*. On the web:
`http://www.vrco.com/CAVE_USER/`.

[250] C. Ware, K. Arthur, and K.S. Booth. Fish tank virtual reality. In *Proceedings of ACM CHI'93 Conference*, pages 37–41, 1993.

[251] K. Watsen and M. Zyda. Bamboo - a portable system for dynamically extensible, real-time, networked, virtual environments. In *IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, Atlanta, Georgia, 1998.

[252] Kent Watsen, Rudolph P. Darken, and Michael V. Capps. A handheld computer as an interaction device to a virtual environment. In *Proceedings of the third Immersive Projection Technology Workshop*, Stuttgart, Germany, May 10-11 1999.

[253] Ascension webpage. `http://www.ascension-tech.com`.

[254] FakeSpace webpage. `http://www.fakespace.com/`.

[255] Intersense webpage. `http://www.isense.com/`.

[256] Northern Digital Inc. webpage. `http://www.ndigital.com/`.

[257] Polhemus webpage. `http://www.polhemus.com/`.

[258] Brent B. Welch. *Practical Programming in Tcl and Tk (second edition)*. Prentice-Hall International, 1997. ISBN 0-13-616830-2.

[259] Matthias M. Wloka and Eliot Greenfield. The virtual tricorder: A uniform interface for virtual reality. In *ACM Symposium on User Interface Software and Technology*, pages 39–40, 1995.

[260] Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE Computational Science and Engineering*, pages 10–21, October-December 1997.

[261] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide (second edition)*. Addison-Wesley, 1996. ISBN 0-201-46138-2.

[262] H. Wright, K.W. Brodlie, and M.J. Brown. The dataflow visualization pipeline as a problem solving environment. In M. Göbel, J. David, P. Slavik, and J.J. van Wijk, editors, *Virtual Environments and Scientific Visualization '96*, pages 267–276. Springer Verlag, New York, NY, 1996.

[263] Lizhong Wu, Sharon L. Oviatt, and Philip R. Cohen. Multimodal integration - a statistical view. *IEEE Transactions on Multimedia*, 1(4):334–341, 1999.

[264] Nicole Yankelovich, Gina-Anne Levow, and Matt Marx. Designing SpeechActs: Issues in speech user interfaces. In *CHI '95 Conference Proceedings*, pages 369–376, May 7-11, 1995.

[265] Christine Youngblut, Rob E. Johnston, Sarah H. Nash, Ruth A. Wienclaw, and Craig A. Will. Review of virtual environment interface technology. Technical report, Institute for Defense Analyses (IDA), Alexandria, VA, 1996. IDA Paper P-3186. On the web: `http://www.hitl.washington.edu/scivw/IDA/`.

[266] E. Kent Yucel, Charles M. Anderson, Robert R. Edelman, Thomas M. Grist, Richard A. Baum, Warren J. Manning, Antonio Culebras, and William Pearce. Magnetic resonance angiography (update on applications for extracranial arteries). *Circulation*, 100:2284–2301, 1999.

[267] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. System integration for interactive simulation systems using intelligent agents. In R.L. Lagendijk, J.W.J. Heijnsdijk, A.D. Pimentel, and M.H.F. Wilkinson, editors, *Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging, Heijen, the Netherlands*, pages 399–406, Delft, May 30-June 1 2001. Advanced School for Computing and Imaging (ASCI). ISBN 90-803086-6-8.

[268] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. AG-IVE: an agent based solution to constructing interactive simulation systems. In P.M.A. Sloot, C.J. Kenneth Tan, Jack J. Dongarra, and Alfons G. Hoekstra, editors, *International Conference on Computational Science (ICCS), Amsterdam, the Netherlands*, volume 2329 of *Lecture Notes in Computer Science (LNCS)*, pages 693–703, Berlin, April 2002. Springer-Verlag. ISBN 3-540-43594-8.

[269] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. Scenario switches and state updates in an agent-based solution to constructing interactive simulation systems. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, pages 3–10, January 2002.

[270] Zhiming Zhao. Note on the use of TAO in HLA. Private communication.

[271] Frans W. Zonneveld. A decade of clinical three-dimensional imaging: A review. part III. image analysis and interaction, display options, and physical models. *Investigative Radiology*, 29(5):716–725, 1994.

[272] Frans W. Zonneveld and Keizo Fukuta. A decade of clinical three-dimensional imaging: A review. part II. clinical applications. *Investigative Radiology*, 29(5):574–589, 1994.

[273] K. J. Zuiderveld, A. H. Koning, R. Stokking, J. A. Maintz, F. Appelman, and M. A. Viergever. Multimodality visualization of medical volume data – our techniques, applications, and experiences. *Computer and Graphics*, 20:775–791, December 1996.

[274] Karel J. Zuiderveld. VR in radiology – first experiences at University Hospital Utrecht. *Computer Graphics*, pages 47–48, November 1996.

Samenvatting

(Summary in Dutch)

Computers zijn over de afgelopen jaren steeds krachtiger geworden. Als gevolg daarvan zijn onderzoekers in staat om steeds grotere en ingewikkeldere problemen te bestuderen. Die toename heeft er echter toe geleid dat de hoeveelheid en complexiteit van de gegevens die door die problemen gegenereerd worden, eveneens toenemen. Vaak blijkt het erg lastig te zijn om die gegevens door een computer te laten analyseren, enerzijds omdat we de computer niet kunnen uitleggen wat interessant is en wat niet, anderzijds omdat de computer er veel te lang over zou doen om een resultaat te vinden. In die gevallen kan het zin hebben om de onderzoeker samen met de computer op ontdekkingsreis te sturen om een beter inzicht te krijgen. In dit proefschrift wordt een onderscheid gemaakt tussen statische en dynamische ontdekkingsomgevingen. In het eerst geval zijn de gegevens al eens op een eerder tijdstip door een computer programma berekend en zullen dus niet veranderen tijdens de ontdekkingsreis. In het tweede geval worden de gegevens berekend door een programma dat loopt *tijdens* de ontdekkingsreis waardoor de gegevens continu veranderen. Die laatste categorie maakt het mogelijk om het gedrag van een lopend programma niet alleen te bestuderen, maar ook te wijzigen.

Om zo'n ontdekkingsreis mogelijk te maken, moeten de te bestuderen gegevens in de allereerste plaats op een begrijpelijke en zorgvuldige manier aan de onderzoeker gepresenteerd worden. Binnen de wetenschappelijke visualisatie zijn verschillende methoden ontwikkeld om dat mogelijk te maken, maar in veel gevallen zijn de gegevens zó complex, dat het nog steeds moeilijk is om inzicht te krijgen aan de hand van een stilstaand plaatje op een beeldscherm. In dergelijke gevallen kan een "virtuele werkelijkheid" (Virtual Reality, VR) uitkomst bieden.

Met een virtuele omgeving (Virtual Environment, VE; de kunstmatige wereld die men kan waarnemen met een VR systeem) wordt geprobeerd om gegevens op dusdanige manier aan de onderzoeker te presenteren dat deze het gevoel krijgt ondergedompeld te worden in zijn gegevens. De gegevens worden daartoe weergegeven als virtuele objecten in een door de computer gegenereerde kunstmatige wereld. Door deze objecten zich te laten gedragen alsof het echte objecten zijn, is de onderzoeker in staat de gegevens te manipuleren zoals hij dat gewend is uit de echte wereld. Op deze manier wordt de gebruikelijke barrière, bestaande uit een plat beeldscherm, toetsenbord en muis, doorbroken en is de onderzoeker meer betrokken bij zijn gegevens.

Om een beter inzicht te krijgen in de technologische aspecten, de toepassingsgebieden

en de wetenschappelijke implicaties die van belang zijn bij de totstandkoming van een bruikbare omgeving, is een drietal statische omgevingen gebouwd (hoofdstuk 2). Daarbij bleek dat de technologie die ons ter beschikking stond wezenlijke beperkingen had die productief gebruik in de weg staan. Uit de ervaringen die zijn opgedaan met deze omgevingen bleek wél dat het gebruik van een VE in sommige toepassingsgebieden inderdaad kan helpen bij het inzichtelijk maken van grote hoeveelheden complexe gegevens.

Tot nu toe betekende het gebruik van VR dat er apparatuur aangeschaft moest worden die voor velen onbetaalbaar was. Eind jaren negentig waren de prijs en prestaties van de huis-tuin-en-keuken personal computer (PC) echter dusdanig, dat het in principe mogelijk was om een goedkoop VR systeem te bouwen op basis van onderdelen die in de winkel verkrijgbaar zijn. Hoofdstuk 3 beschrijft hoe een dergelijk systeem opgebouwd kan worden en laat zien dat de prestaties van een PC gebaseerd VR systeem vrij goed zijn als we die vergelijken met een commercieel verkrijgbare oplossing.

Om bruikbare ontdekkingsomgevingen te kunnen maken voor wetenschappelijk onderzoek, bleek dat sommige essentiële bouwstenen niet voor het grijpen lagen. Dat gold met name voor de wetenschappelijke representatie van gegevens in de vorm van virtuele objecten, de manipulatie van die virtuele objecten, de mogelijkheid om te kunnen meten aan virtuele objecten en de interactie met een virtuele omgeving over het algemeen. In hoofdstuk 4 beschrijven we een aantal technieken en methoden die ontwikkeld zijn om hiervoor een oplossing te bieden.

Een dynamische omgeving kan baat hebben bij een ontwerp waarbij het computer programma dat we willen bestuderen gescheiden wordt van het deel dat de presentatie van gegevens voor zijn rekening neemt en het deel van waaruit we de ontdekkingsreis coördineren. Op die manier is het bijvoorbeeld mogelijk om speciale computer apparatuur in te zetten voor delen die sneller op dergelijke apparatuur kunnen draaien. De consequentie is dan wél dat we maatregelen moeten nemen om de verschillende delen door middel van een computernetwerk aan elkaar te knopen. Als dat onzorgvuldig gebeurd, dan zou het zo kunnen zijn dat de winst die we dachten te halen door de verschillende delen te distribueren over snelle computer apparatuur, teniet gaat door de tijdsvertraging als gevolg van communicatie tussen de delen. Hoofdstuk 5 beschrijft een aantal bestaande oplossingen om dergelijke gedistribueerde ontdekkingsomgevingen te kunnen bouwen. Dit hoofdstuk beschrijft tevens een drietal methoden waarmee de capaciteit van een netwerkverbinding zo efficiënt mogelijk benut kan worden.

Om alle voorgaande technieken en methoden te testen is een prototype gebouwd van een dynamische ontdekkingsomgeving, toegepast op een medische vraagstelling. In deze omgeving is een vloeistofstroming simulatie gecombineerd met een interactieve virtuele omgeving waarin de resultaten van de simulatie aan de onderzoeker gepresenteerd worden. In dit specifieke geval heeft de omgeving als doel om een chirurg te assisteren in het nemen van een beslissing met betrekking tot de meest optimale procedure om de bloeddoorstroming in een patiënt met een vaatverwijding (een aneurysma) of -vernauwing (een stenose) te herstellen. Uit eerste ervaringen blijkt

deze omgeving redelijk te presteren. Het is echter duidelijk dat er nog een lange weg vóór ons ligt voordat dit systeem in realistische situaties gebruikt zou kunnen worden. Momenteel werken verschillende onderzoekers binnen de Sectie Computational Science aan deze toepassing, op diverse onderzoeksgebieden.

Publications

[1] R.G. Belleman, P.M.A. Sloot, and L.O. Hertzberger. Multivariate data processing system: transputer based data acquisition, analysis and presentation. In *Parallel Computing and Transputer Applications (PACTA 92)*, pages 1147–1155. IOS Press, Amsterdam, Washington, September 1992.

[2] R.G. Belleman. CACE - a case study in embedded system design. Master's thesis, University of Amsterdam, September 1997.

[3] R.G. Belleman, J.A. Kaandorp, and P.M.A. Sloot. A virtual environment for the exploration of diffusion and flow phenomena in complex geometries. *Future Generation Computer Systems*, 14(3-4):209–214, 1998.

[4] R.G. Belleman, J.A. Kaandorp, and P.M.A. Sloot. Interactive environments for the exploration of large data sets. In B.M. ter Haar Romeny, D.H.J. Epema, J.F.M. Tonino, and A.A. Wolters, editors, *Proceedings of the fourth annual conference of the Advanced School for Computing and Imaging, Lommel, Belgium*, pages 264–268, Delft, June 9-11 1998. Advanced School for Computing and Imaging (ASCI).

[5] R.G. Belleman, J.A. Kaandorp, D. Dijkman, and P.M.A. Sloot. GEOPROVE: Geometric probes for virtual environments. In P.M.A. Sloot, M. Bubak, A. Hoekstra, and L.O. Hertzberger, editors, *High Performance Computing and Networking (HPCN'99), Amsterdam, the Netherlands*, number 1593 in Lecture Notes in Computer Science, pages 817–827, Berlin, April 1999. Springer-Verlag. ISBN 3-540-65821-1.

[6] R.G. Belleman, J.A. Kaandorp, D. Dijkman, and P.M.A. Sloot. GEOPROVE: Geometric probes for virtual environments. In M. Boasson, J.A. Kaandorp, J.F.M. Tonino, and M.G. Vosselman, editors, *Proceedings of the fifth annual conference of the Advanced School for Computing and Imaging, Heijen, the Netherlands*, pages 38–43, Delft, June 15-17 1999. Advanced School for Computing and Imaging (ASCI).

[7] R.G. Belleman, Z. Zhao, G.D. van Albada, and P.M.A. Sloot. Design considerations for the construction of immersive dynamic exploration environments. In L.J. van Vliet, J.W.J. Heijnsdijk, T. Kielmann, and P.M.W. Knijnenburg, editors, *Proceedings of the sixth annual conference of the Advanced School for Computing and*

Imaging, pages 195–201, Delft, June 14-16 2000. Advanced School for Computing and Imaging (ASCI). ISBN 90-803086-5-x.

[8] R.G. Belleman and P.M.A. Sloot. The design of dynamic exploration environments for computational steering simulations. In Marian Bubak, Jacek Mościński, and Marian Noga, editors, *Proceedings of the SGI Users' Conference*, pages 57–74, Kraków, Poland, October 2000. Academic Computer Centre CYFRONET AGH. ISBN 83-902363-9-7.

[9] R.G. Belleman, B. Stolk, and R. de Vries. Immersive virtual reality on commodity hardware. In R.L. Lagendijk, J.W.J. Heijnsdijk, A.D. Pimentel, and M.H.F. Wilkinson, editors, *Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging, Heijen, the Netherlands*, pages 297–304, Delft, May 30-June 1 2001. Advanced School for Computing and Imaging (ASCI). ISBN 90-803086-6-8.

[10] R.G. Belleman and P.M.A. Sloot. Simulated vascular reconstruction in a virtual operating theatre. In H.U. Lemke, M.W. Vannier, K. Inamura, A.G. Farman, and K.Doi, editors, *15th International Congress and Exhibition, Computer Assisted Radiology and Surgery (CARS 2001)*, number 1230 in Excerpta Medica, International Congress Series, pages 938–944, Amsterdam, the Netherlands, June 2001. Elsevier Science B.V. ISBN 0-444-50866-X.

[11] R.G. Belleman and R. Shulakov. High performance distributed simulation for interactive simulated vascular reconstruction. In P.M.A. Sloot, C.J. Kenneth Tan, Jack J. Dongarra, and Alfons G. Hoekstra, editors, *International Conference on Computational Science (ICCS), Amsterdam, the Netherlands*, volume 2331 of *Lecture Notes in Computer Science (LNCS)*, pages 265–274, Berlin, April 2002. Springer-Verlag. ISBN 3-540-43594-8.

[12] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. System integration for interactive simulation systems using intelligent agents. In R.L. Lagendijk, J.W.J. Heijnsdijk, A.D. Pimentel, and M.H.F. Wilkinson, editors, *Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging, Heijen, the Netherlands*, pages 399–406, Delft, May 30-June 1 2001. Advanced School for Computing and Imaging (ASCI). ISBN 90-803086-6-8.

[13] H. Afsarmanesh, R.G. Belleman, A.S.Z. Belloum, A. Benabdelkader, J.F.J. van den Brand, G.B. Eijkel, A. Frenkel, C. Garita, D.L. Groep, R.M.A. Heeren, Z.W. Hendrikse, L.O. Hertzberger, J.A. Kaandorp, E.C. Kaletas, V. Korkhov, C.T.A.M. de Laat, P.M.A. Sloot, D.Vasunin, A. Visser, and H.H. Yakali. VLAM-G: A grid-based virtual laboratory. *Scientific Programming (Special issue on Grid Computing)*, 10(2):173–181, 2002. ISSN 1058-9244.

[14] Robert Belleman and Peter Sloot. Dynamic exploration environments. In Jeroen Meij, editor, *Dealing with the data flood (mining data, text and multimedia) (STT*

65), pages 771–787. STT/Beweton, The Hague, the Netherlands, 2002. ISBN 90-804496-6-0.

[15] K.A. Iskra, R.G. Belleman, G.D. van Albada, J. Santoso, P.M.A. Sloot, H.E. Bal, H.J.W. Spoelder, and M. Bubak. The polder computing environment: a system for interactive distributed simulation. *Concurrency and Computation: Practice and Experience*, 14:1313–1335, 2002.

[16] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. Scenario switches and state updates in an agent-based solution to constructing interactive simulation systems. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, pages 3–10, January 2002.

[17] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. AG-IVE: an agent based solution to constructing interactive simulation systems. In P.M.A. Sloot, C.J. Kenneth Tan, Jack J. Dongarra, and Alfons G. Hoekstra, editors, *International Conference on Computational Science (ICCS), Amsterdam, the Netherlands*, volume 2329 of *Lecture Notes in Computer Science (LNCS)*, pages 693–703, Berlin, April 2002. Springer-Verlag. ISBN 3-540-43594-8.

[18] E.V. Zudilova, P.M.A. Sloot, and R.G. Belleman. A multi-modal interface for an interactive simulated vascular reconstruction system. In *Fourth IEEE International Conference on Multimodal Interfaces (ICMI '02)*, pages 313–318, Pittsburgh, Pennsylvania, 14-16 October 2002. IEEE Computer Society.