



CFD Visualization in Virtual Reality - The Titanic resurrected

Michael ten Caat



Institute of Mathematics
and Computing Science

RUG



Master's Thesis

CFD Visualization in Virtual Reality - The Titanic resurrected

Michael ten Caat

Supervisor:

Prof.dr. A.E.P. Veldman

Institute of Mathematics and Computing Science

University of Groningen

P.O. Box 800

9700 AV Groningen

August 2003

Contents

1	Introduction	3
2	Vision	5
2.1	Human Vision	5
2.2	The Creation of Virtual Reality	6
2.3	Application of Virtual Reality	9
2.3.1	Application Fields	9
2.3.2	Benefits of Virtual Reality	10
2.4	Vision on Vision	12
3	Preparation	15
3.1	Project Description	15
3.2	Evaluation of Visualization Software	15
3.2.1	File Format	16
3.2.2	Scientific Visualization Systems	16
4	Pilot Project	19
4.1	Requirements	20
4.2	Methodology	21
4.2.1	COMFLO	22
4.2.2	VTK	22
4.2.3	VTK to VRML	23
4.2.4	VRMLVIEW	24
4.3	Summary	27
5	Performance	29
5.1	Performance Optimization	29
5.2	Requirements and Capabilities	33
5.2.1	Capabilities	33
5.2.2	Requirements	34
6	Conclusions	37
A	VTK Related Files	39
A.1	Data File	39
A.2	VTK Elements	43
A.2.1	General Visualization Elements	44

A.2.2	Actors	47
A.2.3	Other VTK Features	56
A.2.4	Summary of Quantitative Aspects	59
B	VRMLVIEW	61
B.1	Optional c-shell Script	61
B.2	Movie Script	62
B.3	VRMLVIEW Manual	64

Chapter 1

Introduction

Nowadays, the flow of fluids, liquid or gas, can be simulated by a computer program. This relatively young research area is referred to as Computational Fluid Dynamics (CFD). Within the research program of Computational Mechanics and Numerical Mathematics of the University of Groningen, a CFD computer simulation program, called COMFLO, is continuously developed. Due to the huge amount of CFD data generated, it is important for both developers and users of CFD computer programs to visualize these data. Considering the visualization, we prefer a graphical one. Although ‘graphical visualization’ seems at first sight to be a pleonasm to some people, it can certainly be distinguished from a mental or a verbal visualization. A mental idea is not easily transferable from one person to another, since “individuals uniquely process and encode the information with which they construct their mental images” [9]. And while a verbal depiction is easily transferable, it could be misinterpreted, for example caused by ambiguities or nuance differences. A graphical visualization is, contrary to mental and verbal ones, especially suitable to support the clear transfer of exactly the same image to a multiple audience. Whatever can happen to the interpretation of a graphical visualization is discussed in section 2.4. Unless stated otherwise, visualization in this thesis refers to a graphical one.

Until the start of this project, the visualization of three-dimensional fluid flow data had been restricted to a two-dimensional monitor screen. Recently, as an extra possibility for a three-dimensional visualization in Virtual Reality (VR), a Reality Centre has become available at the University of Groningen. The objective of this thesis is the development of a method for the visualization of CFD data in Virtual Reality, to be controlled by CFD researchers. Since these researchers prefer to concentrate on their CFD related work, visualization activities should remain as straightforward as possible.

The focus of this visualization project is, as stated above, on the CFD computer program COMFLO. Among the several application areas of COMFLO currently are

- *green water flow* [7], *i.e.* the flow of a wave (green due to the presence of phytoplankton) over the deck of a ship, referred to as the *Titanic*, that causes a certain load on structures on the deck of the ship,
- *air curtains* [6] that provide the possibility of combining open doors with climate separation simultaneously (known from shop entrances and cold stores), and

- *the sedimentation of a solid substance on the wall of blood vessels* [10], which is also known as atherosclerosis.

In CFD the creation of *iso-surfaces* is an important technique. Iso-surfaces resemble surfaces on which a certain property has a specific value. Some of them represent surfaces in real life visible to the human eye, such as a water surface or a solid surface, for instance representing a ship. On the other hand, physical quantities normally not visible to the human eye might be of interest, such as temperature or pressure. In green water simulations typically most important is the pressure, a quantity in continuous interaction with velocities. In the world of air curtains obviously temperature is of main interest, but an agreeable temperature should not be acquired at the cost of an unpleasant windy environment. The concentration of the substance causing atherosclerosis is on the other hand most relevant in the origination of this phenomenon.

In this thesis, for starters, general biological principles of vision are introduced in chapter 2. Furthermore in this chapter, general information on the hardware side of VR is included and the value of VR to different professions. At the end of the chapter a couple of psychological benefits and a quick reflection on our acceptance of VR are shown. Chapter 3, in which software aspects are evaluated, serves as a preparation for the actual project. In chapter 4 the methodology of our pilot CFD visualization is presented as a standard for future CFD presentations at the University of Groningen. In chapter 5 thereafter, visualization demands and hardware capacities are related to each other. Solutions to possible conflicts in this relationship are also brought up. Finally in chapter 6 conclusions are put together.

The details within the visualization process are explained in appendix A by discerning a framework and arbitrary elements that can be included in it. The results of our operation on the CFD data, exported to a series of three-dimensional images, is handled by a viewer that is documented in appendix B.

Chapter 2

Vision

This chapter is a short treatment of vision in general and includes fundamental biological principles on which visualization in general is based, in particular in Virtual Reality. Information is provided on the hardware aspect of our Reality Centre and the value for several groups of persons. It is concluded by presenting a few general psychological benefits of VR and a short philosophical discussion of our acceptance of virtual images.

2.1 Human Vision

In this section, we do not tend to give an extensive description of human vision but just touch some interesting and relevant parts. It is briefly described how humans perceive images and what is the basis of stereo vision.

Human vision is based on the perception of color and brightness. Of what we see, an image is produced on the retina at the back of our eye. This retina is covered with sensitive nerve cells called photoreceptors, responsible for passing the stimulus caused by the image to our brain. The photoreceptors consist of rods and cones. Of the cones, three types exist, of which each type is sensitive to different wavelengths, thus to different colors. The rods are more sensitive than cones, however not to colors, and take care of brightness.

Three important terms in the context of human vision related to the photoreceptors and their functions are [5, 13]:

- *value*, that represents the brightness or intensity,
- *hue*, which reflects the dominant wavelength, thus the dominant color, and
- *saturation*, or chroma, that defines the purity of the color.

Together in the HSV (**H**ue, **S**aturation, **V**alue) color model, these three items mimic the way humans perceive color. Another common model based on the **r**ed, **g**reen, and **b**lue intensities is the RGB color model.

From the images projected on both of our retinae, beside color and brightness we can also perceive depth, referred to as *stereo vision*. This is caused by the slight difference in the angle of view between both of our eyes, which results in a slight difference in both two-dimensional

images. This difference in images causes a three-dimensional perception in our mind, referred to as *binocular parallax* [13].

The main advantage of the visualization of data in general is the ability to make use of the natural abilities of the human vision system [13]. More than half of the total amount of our neurons is actually supposed to be devoted to vision. Beside the binocular parallax that enables us to naturally integrate different viewpoints and other visual clues into a mental image of a 3D object or plot, we have strong two-dimensional visual abilities. In our context, the talent of the human mind estimated to be the most relevant is the one for the recognition of temporal changes in an image, either two- or three-dimensional. Without any effort we recognize trends and spot areas of rapid change in a large series of frames.

2.2 The Creation of Virtual Reality

This section is to explain the production of three-dimensional images in Virtual Reality. All systems that produce three-dimensional images for VR purposes, also referred to as *stereoscopic* or *stereo* images, offer slightly different images to both of our eyes. In this way, the existence of the binocular parallax is made use of to imitate a three-dimensional image. This three-dimensional perception of an image is sometimes referred to as an *illusion*. In section 2.4 we will reflect on this terminology. Two techniques to create stereo images that can be distinguished are [13]

- a *time-multiplexed* technique that alternates between left and right eye images, and
- a *time-parallel* technique that displays both images at once in combination with a process to extract left and right eye views.

Time-multiplexed techniques are typically combined with methods to project alternating images. Liquid crystal *shutter glasses* offer the possibility to view the image with the eye corresponding to the image. Both its glasses are alternately transparent or opaque.

One time-parallel technique converts RGB values, thus color information, into intensity and is called red-blue (or red-green or red-cyan) stereo. The left eye can only see the image through a red filter, the right eye only through a blue filter. A second time-parallel technique preserves the color information from the original images. It uses polarized light. Images projected through a vertical polarizing filter can only be viewed through a vertical filter, while horizontal filters are applied analogously.

Below, the Virtual Reality installations at the University of Groningen are described.

The Reality Cube and the Reality Theatre

In the Reality Centre of the University of Groningen, both a Reality Cube and a Reality Theatre have been present since September 2002. Both VR facilities can be found in the Centre for High Performance Computing and Visualization (HPC/V, www.rug.nl/rc/hpcv).

The Reality Theatre has a curved screen on which stereo images are projected by three projectors in front of it above the audience. An overview of the VR room with the Reality

Theatre is displayed in figure 2.1 (taken from www.rug.nl/rc/hpcv).

The Reality Cube is a CAVE-like VR installation. It is a cube of $2.5\text{ m} \times 2.5\text{ m} \times 2.5\text{ m}$ provided with four combinations of projectors and mirrors. Three combinations are positioned on the outside right next to the cube and project their stereoscopic images onto its three sides, whereas one combination projects on the floor from above. An overview of the location of mirrors and projectors for a CAVE-like VR installation is shown in figure 2.2 (taken from www.rug.nl/rc/hpcv). The Reality Cube is the second of its kind in the Netherlands, the CAVE of SARA (www.sara.nl) is the first.

Both VR installations in Groningen use a time-multiplexed technique, alternating between left and right eye images. Images produced can be viewed in both cases with Liquid Crystal Shutter Glasses, that switch opacity alternating between the left and the right eye. Additional to the shutter glasses in the Reality Cube is a magnetic tracking device to provide information on the position of the viewer and the viewing direction. Also present in the Reality Cube is a programmable sort of three-dimensional mouse, the so-called *wand*. This wand is also tracked for its position and orientation. It has got three normal mouse buttons and a rubber ball as a kind of joy stick to control two continuous values.

Technical Aspects

In the Reality Cube, as a user moves his or her head, images are adjusted to the point of view. A powerful system to provide the images is the SGI ONYX 3400 present at the HPC/V. For each of the four projection planes two images are computed repeatedly. To get the feeling of being immersed, these stereo images have to be computed and generated in *real time*. To contribute to a smooth perception, the minimal required frame rate, the number of computed images to project per second, is 10 Hz. Meanwhile the refresh frequency of the computed images should be at least 25 Hz for a realistic effect without perceiving the alternation between the left and right eye projections. For the ONYX this refresh rate is always 96 Hz, such that this is further no point of attention. To accomplish the required frame rate of 10 Hz however, the complexity of the virtual environment is somehow to stay within the limits of the technical possibilities of the VR installation.

Globally, the technical restriction of the visualization hardware is not considered as the most severe problem. Most serious is in general the complexity to analyze and interpret the enormous amount of data already available or that will be generated in the future. Even modern computers can provide us much more results than we can properly analyze completely visually. By Van Dam *et al.* [4] Artificial Intelligence, a field interested in the working of the human mind in relation to technology, is supposed to provide computer techniques to acquire insight in data. Whether such a computer technique as a product of Artificial Intelligence can be distinguished from a product of ordinary logical thinking can be questioned. Such a translation of human capacities to form a basis for a computer program can after all also be considered as an extrapolation of human thinking.

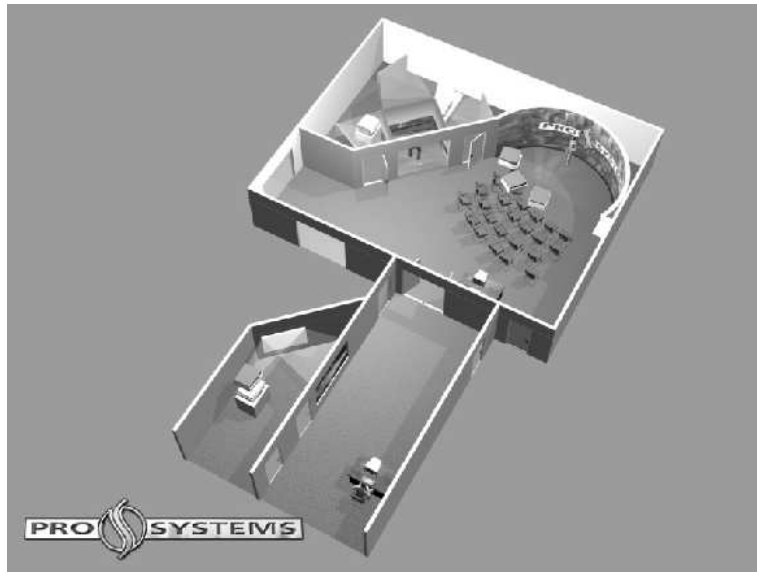


Figure 2.1: *The positioning of the projectors in the Reality Theatre above the audience in front of the curved screen.*

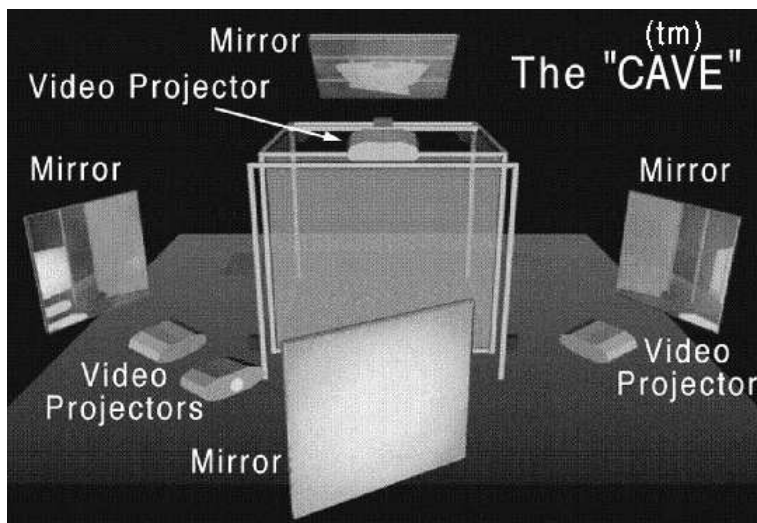


Figure 2.2: *Locations of mirrors and projectors with respect to the Reality Cube.*

Some technical information on the ONYX 3400:

- 4 graphic pipes,
- 512 Mb texture memory per pipe,
- 16 IP35 processors of 500 MHz,
- main memory size: 20480 Mb,
- CPU instruction cache size: 32 Kb,

- CPU data cache size: 32 Kb, and
- 500 Gb disk space.

2.3 Application of Virtual Reality

In this section some application fields of VR and benefits of different visualization methods are presented.

2.3.1 Application Fields

One application in which VR can be involved is *computer art*, which is partly an expression of fantasy. However, for most people, among whom scientists, reality is often sufficiently fantastic. Below, we focus on more practical applications than art. In those practical applications we can distinguish between the visualization of objects already present in reality and objects to become present in reality.

Objects to visualize that are to become reality can be buildings or industrial products, for instance cars. To imagine how the environment present would be affected by a particular new *architecture*, VR could be a very helpful assistant. In areas of technological innovation, a new *industrial design* can be spatially visualized and can be easily adjusted if necessary. So you can have a seat in a new car and look around in its interior and have items replaced even before it is built.

Other areas visualize data and situations already present in today's world. A well-known area is *medical visualization*. Nowadays, various kinds of scans with a medical application are made, like MRI- (Magnetic Resonance Imaging) or CT-scans (Computed Tomography, also known as CAT-scans which stands for Computer Assisted Tomography). These scans result in series of two-dimensional 'slices' of the human body. From this series of two-dimensional images, a three-dimensional reconstruction is made. Although radiologists are trained to deal with this series of two-dimensional images, a three-dimensional reconstruction makes it possible for a surgeon to acquire the three-dimensional view he is accustomed to in practice. In the case of neuroscientific data, the task of organizing, analyzing and presenting data is referred to as *neuroinformatics* [12].

A completely different semi-medical application, recently described by Bohannon [2], is the three-dimensional visualization of *human genome* data. The mountains of human genome data available can result in an unclear view, while in three dimensions a better understanding is acquired. A comparison of gene regions between species, *e.g.* between the human species and laboratory animals, is also made much easier, creating opportunities for *drug discovery*.

In *psychiatric* and *psychological* applications, usually real life situations are simulated. (For now we distinguish between psychiatric and medical, although psychiatry is of course part of medicine.) In these real life situations, persons are always involved as actors, contrary to for example CFD applications. Promising research areas are anxiety disorders, eating disorders, neuropsychological applications, and distraction techniques for painful medical procedures. These four areas have been successfully incorporated into existing clinical protocols and are

recognized as appropriate adjuncts to clinical practice (Wiederhold [15]). Furthermore, flight or driving simulators are potentially straightforward applications to train or study the human mind to obtain an insight in or to support the spatial learning process (Gamberini [8]).

By chemists VR is applied to get an increased comprehension of the three-dimensional structure of *molecules*. CFD researchers are provided the opportunity to check output data for errors, relying on the natural ability of humans to recognize trends and spot areas of rapid change in a large series of frames (see section 2.1). Meanwhile, a quantitative spatial insight in the three-dimensional data is acquired. To other interested parties, the reliability and capabilities of CFD can be shown in a decorated visualization in which the emphasis is on realism.

2.3.2 Benefits of Virtual Reality

All Virtual Reality installations have the same advantage of a computer generated environment. This environment is also referred to as a Virtual Environment (VE) in which we assume to be present a certain way to generate stereo images. In such a VE, variables such as objects, colors, and light can be easily manipulated. Besides, the experimental situation is fully controlled and any measurement of time, distance, and performance is possible. The special benefit of a VR installation in which the user is being tracked, like the CAVE-like Reality Cube, is the feeling of being fully immersed. Whereas the Reality Theatre lacks this particular possibility to track a user, this installation with its curved screen is more appropriate for presentations for larger groups, up to twenty one persons.

In some applications one may question the use of an expensive Reality Cube, while possibly other VR systems would suffice. A desktop Virtual Environment using the monitor screen of a computer that is somehow provided with stereo images is an example of another VR system, but it has got no possibilities to track a person's head. A system less complicated than a CAVE that can also track the users orientation is a Head Mounted Display (HMD). An HMD exists of a pair of glasses with images for the left and right eye projected on its glasses. If the possibility is present to track a user for his orientation, the application can generate a so-called *immersive* VE. Thus, an HMD and a Cube are immerse Virtual Environments, while desktop VEs and the Reality Theatre are *non-immersive*. The original idea to develop immersive VEs in addition to non-immersive ones is not at all surprising: humans have a lifetime of immersive experience in four-dimensional physical worlds, *i.e.* three-dimensional space varying in time.

Psychological Aspects

We present a few psychological aspects that provide some guidelines on what kind of VE to use. It is to be noted meanwhile, that this is not at all intended to be a complete overview of psychological aspects of Virtual Reality, but this is meant to present just some aspects to account for.

In any case, if anyone doubted the *credibility* of virtual reality in general, research has shown that performances by people in Virtual Environments are usually similar to those obtained in real situations and are thus valuable (Gamberini [8]). Apparently, Virtual Environments

in general can be useful in the simulation of reality.

Psychological aspects studied in the literature are spatial knowledge, object recognition, the feeling of presence, and leadership. Gamberini [8] investigated spatial knowledge and object recognition in an experiment with an immersive and a non-immersive VE. With respect to the performance in *spatial knowledge* tasks, no meaningful differences were noted between the use of an immersive HMD and a non-immersive desktop. Another, but rather unexpected, conclusion was drawn involving another aspect: persons performed better in *object/pattern recognition* tasks in a non-immersive desktop environment than with an immersive HMD. The cause of this unexpected result was thought to be the less natural navigation with an HMD, since participants in the research had to navigate through their virtual environment. The HMD was provided with two buttons to go forward or backward, while in the desktop environment navigation was controlled by use of the keyboard arrows which most people are more accustomed to. Clearly, the sort of navigation can make a difference. Another cause of the rather unexpected result for object/pattern recognition tasks might have been provided by Van Dam *et al.* [4]. Users do not necessarily need to perform better in immersive virtual environments: mistakes that are made in the real world were also made in the immersive VE provided by the HMD, but not in the non-immersive desktop VE. Obviously, further attention has to be paid to connecting research in Virtual Reality with what happens in real life. Besides, we have to realize that an HMD is not fully comparable to a CAVE-type VE, although both offer an immersive experience.

An investigation involving both an immersive CAVE-like Cube and a non-immersive desktop Virtual Environment simultaneously was done by Axelsson *et al.* [1]. They investigated the feeling of (your own) presence, co-presence (of a partner) and leadership in VR. It concerned an experiment existing of a cooperative task to be performed by couples of persons, first in a virtual and later in the real world. The couples had to build one big cube from several smaller cubes. In the experimental virtual environment, the two persons were seeing each other from two different types of VR systems: a desktop environment and a CAVE-like Cube. By the Cube participant, the feeling of *presence* was noted to be stronger than by the one in the desktop environment. Not surprisingly, at the same time the feeling of *co-presence* was as strong in the Cube as in the desktop environment. Probably the immersed Cube participants were more focused on their task than their partners in the non-immersive desktop VE and did not pay much attention to their partners for that reason. Also the Cube participant was agreed by both participants to be more active and contributive, two properties related to *leadership*. And since leadership is believed to be correlated with technological advantage, the Cube can be concluded to be technologically more advanced than a desktop VE. So the Cube has its advantages above a desktop VE involving the feeling of presence and is technologically more advanced, in accordance with expectations.

Conclusion

Apparently, each VR system studied (desktop, HMD, CAVE) tends to have its own properties and is not easily comparable to other systems. The requirements and obviously the means available are expected to be decisive on what system is to be chosen. More extensive research could result in conclusions that are more straightforward in this field of Virtual Reality that has only recently emerged and with its CAVE-like technology that is still not widespread.

To conclude, all Virtual Environments still have some main advantages in common. They can generate easily adaptable and controllable environments that are comparable to reality. A way to distinguish the different VR installations is to inspect the level of immersion. From the possible influence of the method of navigation, it can also be concluded that all aspects involved in a presentation in Virtual Reality have to be carefully examined.

2.4 Vision on Vision

In this thesis, we accept the biological (section 2.1) and the technological sides of vision (section 2.2) as they are. We are planning to fully trust on the natural human abilities and will not dig into details of the relation between technology and the human mind. We rather have that done by Artificial Intelligence. The longterm goal of Artificial Intelligence has been to develop computer systems that could replace humans in certain applications. A lack of progress in this area has led some researchers to view the role of computers as amplifiers and assistants to humans, as is the role in scientific visualization. Thus, somehow scientific visualization is related to Artificial Intelligence, but the attitude of both fields towards computers is different. Although there is certainly not a lack of progress in the visualization field, there still remains a lot to be developed. As long as for example a thesis like this one cannot be explained in a completely graphical way, the field of visualization theoretically maintains its development possibilities.

Although we accept the biological side of human vision as it is, we still might discuss our vision and ask ourselves the next question. Are the things we see really the things we see? Plato already tried to answer this question in his allegory of the cave, to which the recursive acronym CAVE (Cave Automatic Virtual Environment) is a reference. In this allegory in his Republica, book VII, Plato created a dialogue putting words in the mouths of Socrates and Glaucon.

In this dialogue, a situation is sketched in which prisoners are chained in a cave. Behind the prisoners a bright fire flickered, while people were carrying between the prisoners and the fire cut-outs of all-day objects like trees, animals, and hills. The prisoners, chained with their backs towards the fire being unable to look behind them, faced the cave wall in front of them, on which the shadows of the objects appeared. This situation is sketched in figure 2.3.

For the prisoners, the shadows would be their truth. Until one of them was released to see the sun and the real world of trees, animals, and hills. Once the released prisoner had returned, he could not convince the others of what he had seen. Reason for Plato to tell this story was to cast doubt on the interpretation of what we see. We could be the prisoners and the sun could be our fire in the cave. The journey upwards to the real world could then correspond to an ascent into our internal selves, depending on the interpretation.

Apparently, the images we see could be illusions. An example of an image of which we know that it is different from what we see, an illusion, is shown in figure 2.5. If you stare at it for a few seconds, you will get the illusion of grey spots at intersections, while this illusion disappears if you focus on one intersection. It is referred to as the Hermann grid illusion.

Internally, we might be able to find the real truth, but maybe we are not interested in the real

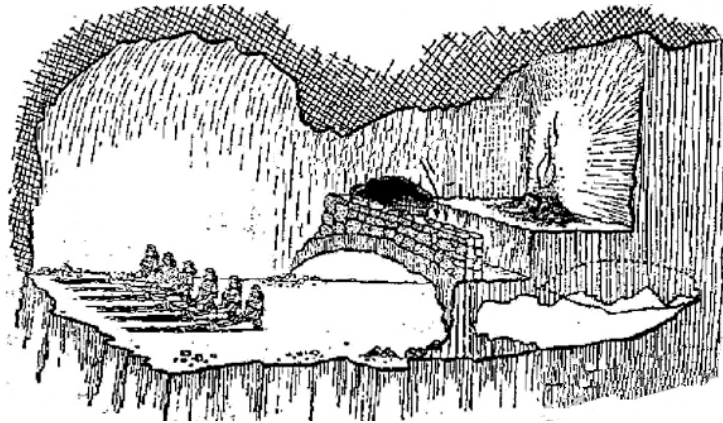


Figure 2.3: *Sketch of Plato's cave, with the chained prisoners facing the left wall, the fire flickering behind them.*

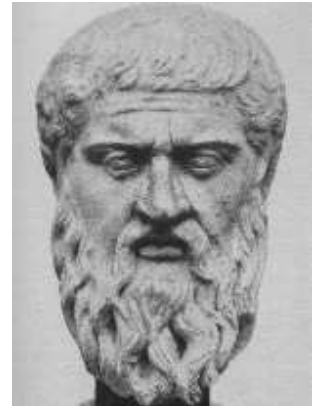


Figure 2.4: *Plato, about 400 B.C.*

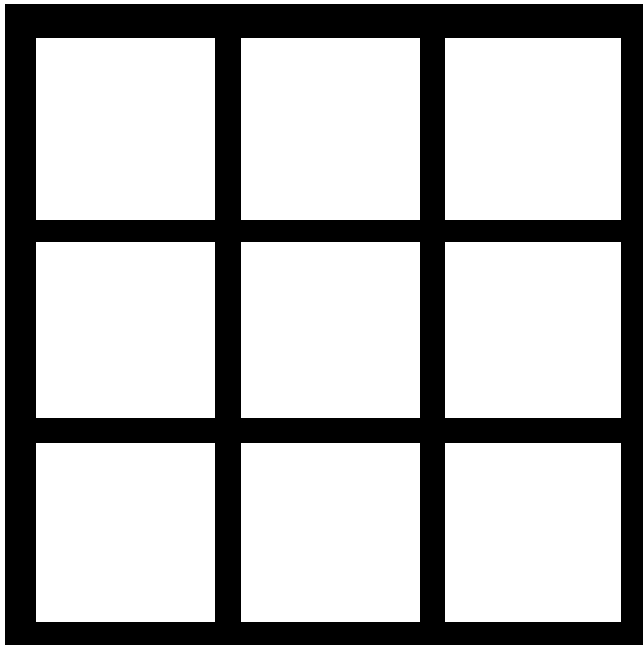


Figure 2.5: *Stare at the image for ten seconds. Then you will get the illusion of grey spots at intersections. If you try to focus on one intersection, this illusion should disappear.*

truth at all. We seem to be content with the current interpretations of what we see, whether they are illusions or not. Main benefit of our satisfaction with illusions instead of real truth is the acceptance of the VR by our mind, whether it is considered to be an illusion or not.

Chapter 3

Preparation

3.1 Project Description

Goal

The goal of this thesis is to describe a way to develop a VR presentation to be controlled by CFD researchers. Whereas Virtual Reality is typically related to visualization, CFD researchers often lack expertise in this field since they have other priorities. For that reason, we try to find a way suitable for the CFD researcher as a casual visualization user.

Starting Point

To reach our goal we start from the sources already present. Concerning knowledge, we strictly approach the project from the CFD researcher's view. Meanwhile, we keep in our minds the presence of different operating systems and accompanying software at the location of the CFD research group and at the location of the VR facilities. Point of attention is the need for the CFD researcher to be able to prepare the presentation at his or her own working space, on a desktop.

At the CFD research group, Linux is the current *operating system*. In addition, the presence of an IRIX operating system by SGI (www.sgi.com) at the VR centre, has to be accounted for, although probably IRIX will be replaced in the future by Linux. In section 3.2, first the *file format* for data storage that is considered to be most appropriate is presented. This is followed by putting together selection criteria for the *visualization system* to be used for the preparation of a visualization in Virtual Reality, and an evaluation of a few of these systems.

3.2 Evaluation of Visualization Software

In this section we try to motivate our choice of the visualization system, to which we also refer as visualization software. In general, a good software design should be robust, understandable, extendable, modular, maintainable, and reusable [13]. Data visualization is a rapidly expanding field, with new visualization techniques being developed each year. As specifications change and grow, a software system needs a solid underlying architecture and design to adapt to expanding requirements.

With these software demands in mind, we investigated the variety of software used at our department and in the literature. Meanwhile, we accounted for the presence of several file formats for the storage of data. We prefer the use of a general and widely used format, appropriate for the presentation of CFD data.

Note: As for the unexperienced visualization user the world of visualization literature seems to be a jungle of acronyms, the use of acronyms is restricted here as much as possible. We try to pursue to use only acronyms that seem to be acknowledged generally and of which the use is thought to be more comfortable in the context.

3.2.1 File Format

In our quest for an appropriate file format, we were restricted to file formats present in Linux visualization software that can function as an input format to a VR presentation in the Reality Cube or Reality Theatre, via an IRIX based system. A file format widely used and generally applicable is preferred.

The *Virtual Reality Modelling Language (VRML)* format meets our requirements. It started as a new 3D Web standard in 1994, and its goal was originally to represent static virtual worlds (VRML 1). Later VRML was adjusted to show interactive three-dimensional worlds, with an animation of series of files as an option (VRML 2). It became a standard and was adopted in the year 1997 as International Standard ISO/IEC 14772-1:1997. Since then it is referred to as VRML97, to which we from now on refer as VRML. For the format, several VRML viewers for various platforms are available (www.web3d.org/vrml/vrml.htm). The original Web oriented character of VRML is still present in its current form. By Coors and Jung [3] VRML is even judged to be 'excellent' for the visualization of three-dimensional geometric information via the WWW.

In the VRML format, adjustments can be made to the visualization as it is a modelling language. However Nielsen and Hansen [12] suggested that VRML should preferably serve as a presentation format, since it has severe limitations as a scientific visualization development environment. For this reason, we try to find visualization software that has integrated object adjustment possibilities, such as texture mapping (the technique to add detail to an image without the requirement of modelling detail, [13]). Further we demand the possibility to export these changes to the VRML format. Only visualization systems with these possibilities are considered in the next section.

Although many computer programs offer the possibility to import files in VRML file format, we do not even need to make use of this. The reason is that a VRML viewer, developed at the VR centre, is available. This viewer is stand-alone and offers the possibility to visualize VRML files on a desktop screen (of a Linux workstation), in the Reality Theater, or in the Reality Cube.

3.2.2 Scientific Visualization Systems

We consider a restricted number of visualization systems, of which it has been assured that they possess the possibilities of texture mapping and export to VRML. They are all available

for several platforms, among which Linux which is used by the CFD developer.

Matlab

The visualization system present at the research group of Computational Mechanics and Numerical Mathematics is **Matlab** (www.mathworks.com). In the past it was preferred after it had been used simultaneously with AVS for a while. In Matlab, for Virtual Reality a special Virtual Reality Toolbox is available. This toolbox requires a license with its accompanying extra costs. Besides, it is linked to Simulink, “an interactive tool for modelling, simulating, and analyzing dynamic, multidomain systems”, to which average CFD researchers are not accustomed. Trivailo [14] experienced that Matlab applications are version-dependent: some older graphical programs could not run under newer versions, which violates our software design demands. Of course, programs developed under newer versions should not necessarily work in older versions, but the other way around it could be useful for older programs to still function in newer versions. In short, the presence of a so-called upward compatibility is preferable.

AVS/Express

A similar argumentation with respect to version dependency influences our opinion on the visualization system AVS or **AVS/Express** (www.avs.com). The newer version AVS/Express appeared to be more complicated to handle than the older version of AVS, especially in developing new extensions to the program. Some more specific information can be found in Dutch via www.rug.nl/rc/hpcv/people/kraak/publications. Since the use of AVS appeared to be not too straightforward, the search for other visualization systems was continued.

VTK

An *open source* programming library devoted to three-dimensional data visualization, designed with extensibility in mind, is the *Visualization Toolkit* (**VTK** [13]). Contrary to previously mentioned experiences with Matlab and AVS(/Express), VTK claims that the addition of new material should not have any significant impact on the existing visualization system. At the department of Computing Science of the University of Groningen, some version dependency of VTK has been noticed.

An evident advantage of VTK over the other two systems mentioned is that it is freely available. By the writers of the VTK book [13], it is said that VTK by its more concrete data model, relative to the more abstract model of AVS, is easier for the casual visualization user. So VTK, based on this argument of users comfort, seems to fit the CFD researcher better than AVS(/Express). However, the presence of a user interface in AVS, contrary to VTK, is not commented on in the book. Still, VTK is a complex program of which the documentation is limited.

The complexity of understanding VTK may be reduced by the following ‘gadget’. For the absence of a graphical user interface in VTK, a solution may be provided by the graphical user interface **MayaVi** (<http://mayavi.sourceforge.net>) written in the programming language Python (www.python.org). This graphical user interface has been and is still being developed especially for CFD data visualization by Ramachandran. At the homepage a description of

MayaVi and its features can be found, as well as a users guide. Technically, it should have no problems running on any platform where VTK and Python are available. Since texture mapping for instance is not (yet) supported by MayaVi, MayaVi is not thought to be the general solution to overcome the complexity of VTK. Still, it is expected to provide some insight into VTK. Dependent on the type of data set, iso-surfaces or streamlines should be easily visualized. However, it does not offer however the possibility to visualize a time series of files, as a result of which it is not suitable to fulfill our desire to create a movie. Anyhow, MayaVi may introduce us to the limited number of VTK features necessary to visualize CFD data.

Conclusion

Summarizing, all visualization systems appear to be accompanied by their characteristic complexities caused by their extensive functionality. To compare the complexities, extensive user experience would be required. Without that extensive user experience we can not judge whether a visualization system is unnecessarily complex. Since obtaining extensive experience would take too much time, a provisional choice for a visualization system is made on basis of arguments provided by other sources than our own experience. Anyhow, it is considered to be part of the project to reduce the complexity of the visualization system of our choice. We plan to do this by restricting the number of features to those features necessary for the visualization of CFD data and by structuring the features involved.

Beside their complexity, all visualization systems evaluated have some version dependency in common. Based on the arguments above considering the users convenience, we have chosen for the visualization system VTK. The complexity of the program is expected to be limited by the application restricted to CFD data, while the graphical user interface MayaVi may assist to get a clearer overview of the limited number of CFD features. Besides, the Visualization Toolkit is a free open source visualization system, involving no license difficulties.

Chapter 4

Pilot Project

To serve as a standard for future CFD visualizations in VR at the University of Groningen, one application is chosen to function as a pilot project. As such a project, the simulation of green water smashing on a structure on the deck of a ship has been selected. Here ‘green’ refers to the color of the water, caused by the presence of phytoplankton. Phytoplankton absorbs red and blue light, and reflects green. In reality also turbulent white water may be observed, however this is not of interest (yet) in our simulations. A photo of the phenomenon



Figure 4.1: *Green water smashing on a ship, accompanied by white water.*

is shown in figure 4.1. This example, showing one of the application areas of COMFLO, is used to illustrate the three purposes for CFD researchers of a presentation in Virtual Reality that are distinguished in section 2.3.1. The primary purpose to the CFD developers is the possibility to inspect the data for anomalies. Secondly, a spatial physical insight reflecting the experiments can be provided by the visualization of a physical quantity of interest. Finally, a presentation to a general public, purely for presentational purposes, could be developed. In

such a case the realism of the results is to be emphasized.

The requirements to accomplish the suggested realistic and quantitative visualizations are put together in section 4.1. The methodology chosen to achieve these requirements is described in section 4.2.

4.1 Requirements

There are a number of design elements we would like to see to be realized in the green water presentation. We first present two elements not explicitly visible that are necessary or useful to assist any visualization.

- To thoroughly observe the several elements of a visualization in general, an arbitrary point of view should be optional. This feature can be made possible by *navigation* and should be controlled in the Reality Cube by the wand, and by keyboard and mouse elsewhere (in the Theatre or at a desktop). To get a universal navigation control, as far as possible, similar movements should be acquired by similar actions via the wand.
- A useful demand is the possibility to *reset* the point of view to the original point of view.

Navigating through our virtual world, we would like to acquire an insight in the spatial distribution of certain physical quantities, meanwhile paying attention to possible anomalies.

- In the set of physical data resulting from our simulations, some aspects normally not visible to the human eye are contained, like pressure. Coloring different pressures differently at several locations provides a *spatial overview* over this physical quantity.

Convinced of the physical realism, we can move away from our quantitative perspective and start to emphasize realism for a presentation to a general public. The remaining elements are objects normally visible when navigating through a real world.

- From our arbitrary point of view we would like to get the feeling of being in a realistic environment. This means being in mid-ocean, in case of the green water simulation. Therefore, beside the simulated wave, a larger number of *waves* stretching out to the *horizon* is desirable.
- Closer to the ship, we would normally see more detail. This detail can be added, without explicitly modelling detail, by the use of *texture mapping*. A texture map on the water surface may also improve the sense of realism. Further, seeing only a bow of a ship floating in mid-ocean does not contribute to the credibility, which can be solved by modelling a complete ship in the presentation.
- As the Reality Cube and Theatre have the opportunity to include *sound* in the presentation, this feature might be used to add the roaring sound of an approaching wave. Since information on the pressure on the deck surface and on the vertical structure on the deck is available, this may be related to the sound volume.

To satisfy the different requirements, a methodology is presented in section 4.2.

4.2 Methodology

To accomplish our task to visualize CFD data, given the requirements of our pilot project presented in section 4.1, we design a methodology that is preferably as uncomplicated as possible. Briefly, we adjust our output data to meet our requirements by the use of the Visualization Toolkit (VTK), and export our changes to VRML files that can be made visible by a VRML viewer, called VRMLVIEW.

A more extensive description of the design of our methodology is as follows. First, COMFLO is adapted to export output data to `vtk`-files, that can function as input to VTK. The data are manipulated by commanding VTK by `tcl`-scripts, adding the specific requirements from section 4.1 one by one. It appears to be convenient to orderly treat different objects, among which the bow of the ship and the moving wave, in separate `tcl`-scripts. We have to translate these properties in VTK to similar properties in VRML code. To visualize the `wr1`-format, as VRML files are identified, a viewer called VRMLVIEW has been developed at the centre for HPC/V. This viewer is able to generate a visualization in either the Reality Cube, the Reality Theatre, or at a desktop screen of a Linux machine. In every environment, a possibility is present via VRMLVIEW that meets our requirements for navigation, as stated in section 4.1, including a reset-option.

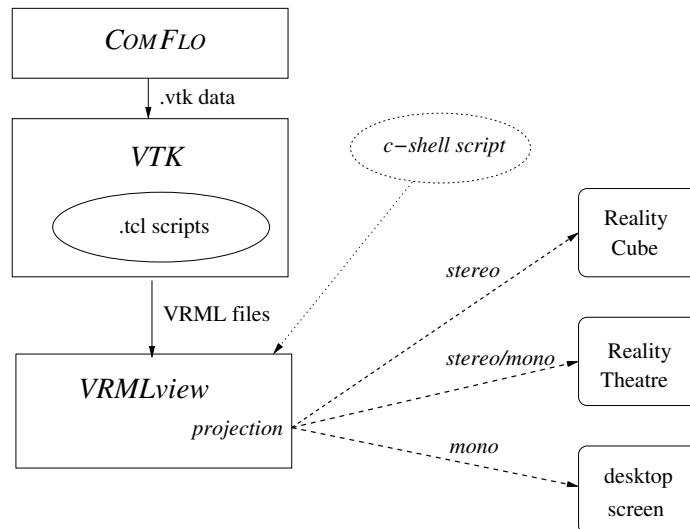


Figure 4.2: *Design of the project methodology. COMFLO output is read by VTK and adjusted, before exporting the visualization to the VRML file format. A VRML viewer presents the VRML files in either the Reality Cube (in stereo), the Reality Theatre (in stereo or mono), or at a Linux desktop (in mono). Additional features, when not supported by VTK, can be easily included by the use of an optional (dotted) c-shell script.*

Of the features of VRML, certain ones are not supported by the VRML exporting routine in VTK (`vtkVRMLExporter`). These particular features can be added ‘manually’ by using a

c-shell script to edit the VRML format. However this script is not obligatory for the visualization to succeed.

Schematically, the design of the methodology is shown in figure 4.2. The additional c-shell script, including extra VRML features and not really necessary to complete the visualization, is surrounded by a dotted ellipse.

In the next sections the separate procedures within our methodology are described more extensively.

4.2.1 COMFLO

Adapting COMFLO to write output to VTK data files is merely adding some writelines in a new routine. In appendix A.1, an example of a standard VTK data file is shown, accompanied by the FORTRAN routine responsible for writing it.

4.2.2 VTK

The way of commanding VTK by `tcl`-scripts may need a more extensive description. For a future user wanting to visualize an application of COMFLO similar to one involving the Titanic visualized in this thesis, it is sufficient to study the existing `tcl`-scripts, explained in appendix A, and make necessary changes. For the future user who wants to do something slightly different or something completely new, the following could be advisory in the use of VTK.

Beside a consultation of the VTK User's Guide (www.vtk.org/buy-books.php), several other opportunities to be informed on the use of VTK exist. Any addition to the visualization via the `tcl`-script starts with some keyword on a subject. Starting with this keyword, it is recommended to first dig for it in VTK documents. These VTK documents consist of standard examples (including `tcl`-scripts) that come with the installation in the directory `/Examples/`, and an extensive documentation of every available option in the file `vtk40DocHtml.tar.gz` (`tar -xzvf` this file), to be found for version 4.0 at www.vtk.org/files/release/4.0/.

If the information provided by the VTK document sources is not sufficient, other ways to reach your goal may be investigated. One other way is provided by the graphical user interface *MayaVi* (introduced in section 3.2.2) that is upon VTK. A simple introduction is contained in the users guide in its installation directory (`doc/guide/book1.html`). This graphical user interface, specially dedicated to CFD purposes, can read `vtk`-data files and can execute several visualization operations, among which the generation of iso-surfaces and streamlines. In *MayaVi* a so-called pipeline browser can be opened to get an overview of all VTK operations related to this specific operation (see figure 4.3). In general, if no suitable `tcl`-script is available to visualize a dataset as desired, *MayaVi* probably offers the easiest opportunity to quickly construct a VTK visualization from a given standard VTK data file. A screenshot of *MayaVi* is shown in figure 4.3.

In some cases, the previous options proposed do not provide a solution. For instance, if one is wondering how or whether a feature in VRML is supported by VTK, an investigation of

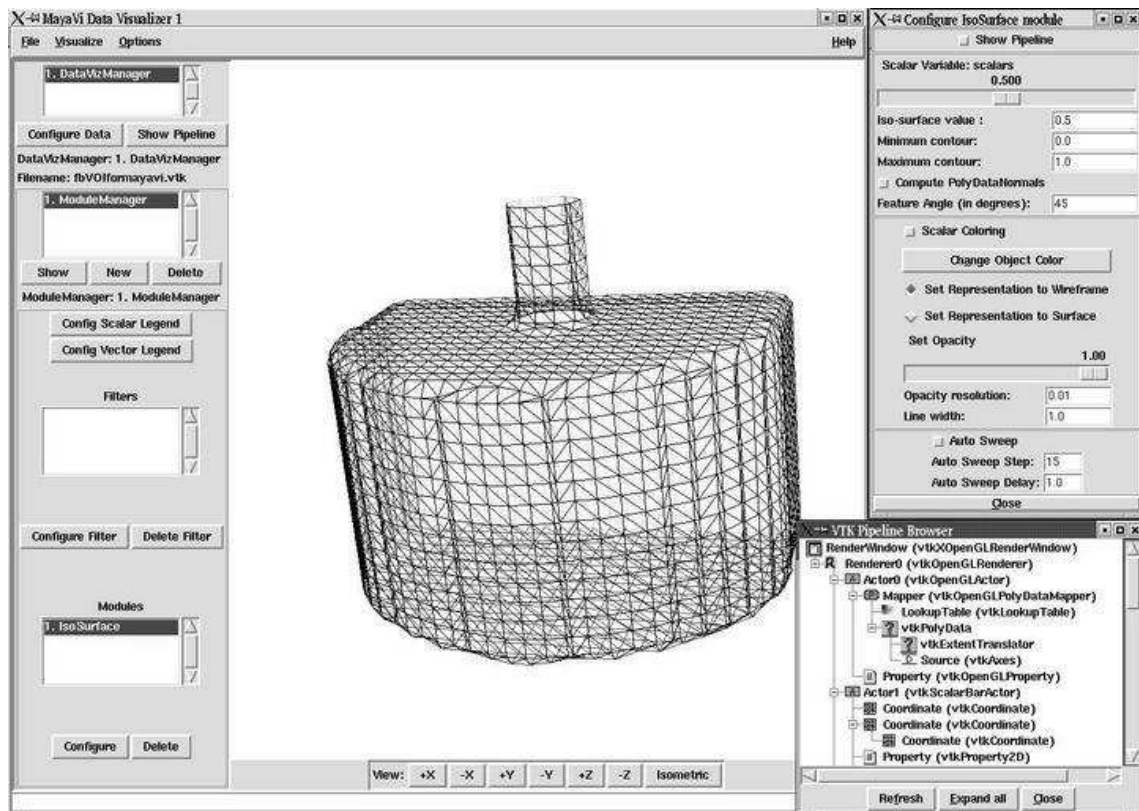


Figure 4.3: A screenshot of *MayaVi*, including a menu to adapt the *IsoSurface* module (upper right) and a *VTK Pipeline Browser* in which an overview over *VTK* elements is shown (lower right). This visualization contains the wire frame representation of the isosurface that represents the bow of a ship.

VRML related files in the source code (in the *VTK* installation directory `/opt/local/vtk/VTK/`) would give some insight in the relation between *VTK* and VRML or would exclude a relation on that topic. Occasionally, without any further inquiries an addition analogous to other elements in the script can easily be made.

Any of the previous suggestions to handle keywords may be expanded by the use of *search engines* on the WWW. Among the query results *tcl*-scripts may be found of which parts can be directly copied into your own *tcl*-script. A possibly useful source of scripts can be found at the *RuGVTK-examples* via (www.rug.nl/rc/hpcv/people/kraak/). During the search for information on keywords, an increasing number of related keywords may be encountered.

4.2.3 VTK to VRML

The results of the operations on the original data set are to be exported to the VRML-format. To carry out this operation with *VTK*, a *VTK* class `vtkVRMLExporter` is available within *VTK*. Several of the characteristics within *VTK* classes coincide with or are related to so-called fields or events in VRML entities called nodes. Some features of VRML however

are not at all supported by the VRML exporter of VTK. These features might be included by the use of a c-shell script (explained in appendix B). Relevant examples of such features in our visualization of the Titanic are `ReflectionMap`, `Navigation type`, and the option `creaseAngle`. `ReflectionMaps` offer you the opportunity to deal with a second visual aspect of water next to its transparent character, namely its reflective character. This aspect appears to contribute to the depth perception, when being in motion or observing a moving object. To be able to smoothly control your moves in the Virtual Environment generated with the VRML viewer, the `Navigation type` is put to `FLY`. To prevent the polygonal faces to be faceted, the crease angle can be given a value greater than or equal to 0. If then the angle between the normals of two adjacent polygons is less than that crease angle, the intermediate polygonal face is shaded smoothly. (More information on VRML is available via www.web3d.org/technicalinfo/specifications/vrml97/vrml97specification.pdf.)

4.2.4 VRMLVIEW

For VRML files a special VRML viewer, developed by B. Hess at the HPC/V, is available. Its use is described in appendix B. Having exported all objects to separate `wr1`-files, they are to be made visible by use of the VRML viewer either at your desktop, in the Reality Cube, or in the Reality Theatre. In order to visualize all fixed and moving objects simultaneously, all separate `wr1`-files containing the different objects are called from one main `wr1`-file. In that main file, the fixed objects and the background are read only once to optimize the performance. The frame rate for the series of files for moving objects can be adapted to the presentation in the main file `wr1`-file.

Navigation with VRMLVIEW on a Desktop and in the Reality Theatre

Although Virtual Reality at a desktop is possible, we lack a possibility for stereo projection at our desktop monitor. However, we can view the mono representation of our visualization. In the Reality Theatre both a mono and a stereo projection are available. Navigation is both at a desktop and at the Theatre to be controlled with the combination of keyboard and mouse. The keyboard controls are schematically shown in figure 4.4. The mouse pointer indicates directions. The change of viewpoint (“**C**” for the previous or “**V**” for the next one) also serves as a reset command.

Navigation with VRMLVIEW in the Reality Cube

In the Reality Cube the wand is available to control navigation. The functions within VRMLVIEW assigned to the three push buttons of it and its ‘joy stick’ are illustrated in figure 4.5.

Screenshots

A screenshot from the movie of the Titanic model in the middle of the ocean, commented on in section 4.3, is shown in figure 4.6. It shows a ship surrounded by a carpet of waves stretching out to the cloudy horizon. In figure 4.7 a purely quantitative visualization is shown. Different pressures are assigned different colors, varying from blue for low to red for high pressures. On the vertical structure you can see at the bottom the difference in color between the centre of the vertical and its edges, even though this is a greyscale image. While the centre is colored

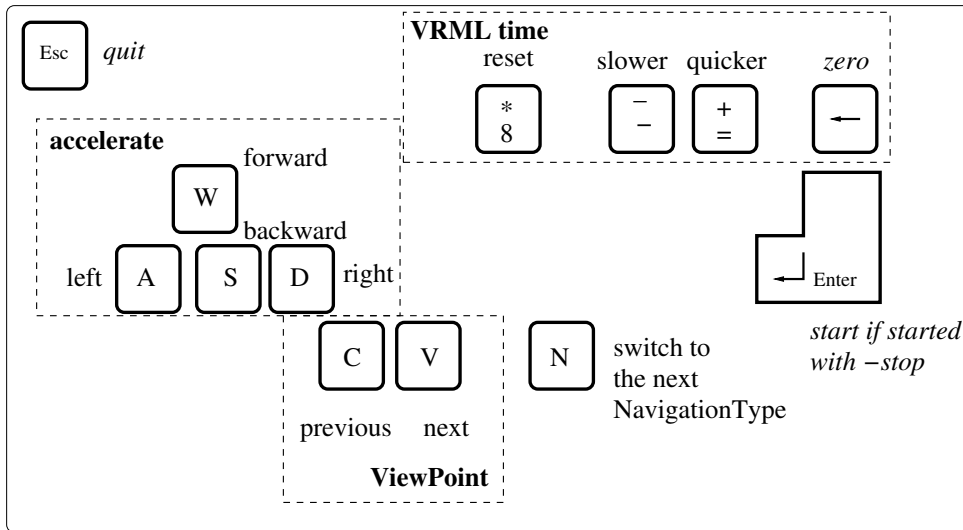


Figure 4.4: The (grouped) keyboard controls of the VRML viewer. All controls are available in both the Reality Theatre and the desktop environment, while in CAVELib mode in the Reality Cube only the controls with descriptions shown in italics are available.

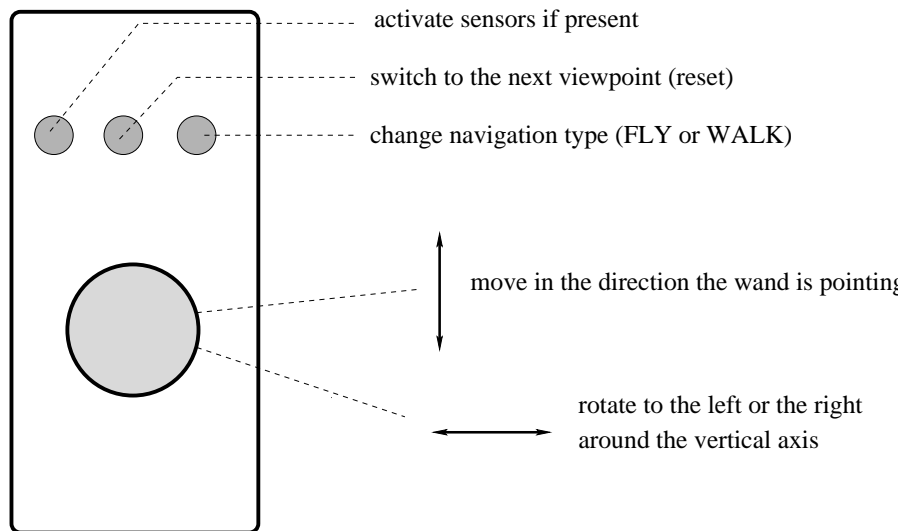


Figure 4.5: Functions attached to the three buttons and joy stick of the wand.

red in the movie, the edges are light blue, indicating a high pressure in the middle and lower pressures around the middle. The darkest color in figure 4.7 coincides with the darkest blue in the movie, representing the lowest (atmospheric) pressure. Note that only the bow is shown in figure 4.7, while in figure 4.6 the complete ship is shown.



Figure 4.6: *The Titanic in mid-ocean, surrounded by waves stretching out to the cloudy horizon (VRMLVIEW screenshot).*

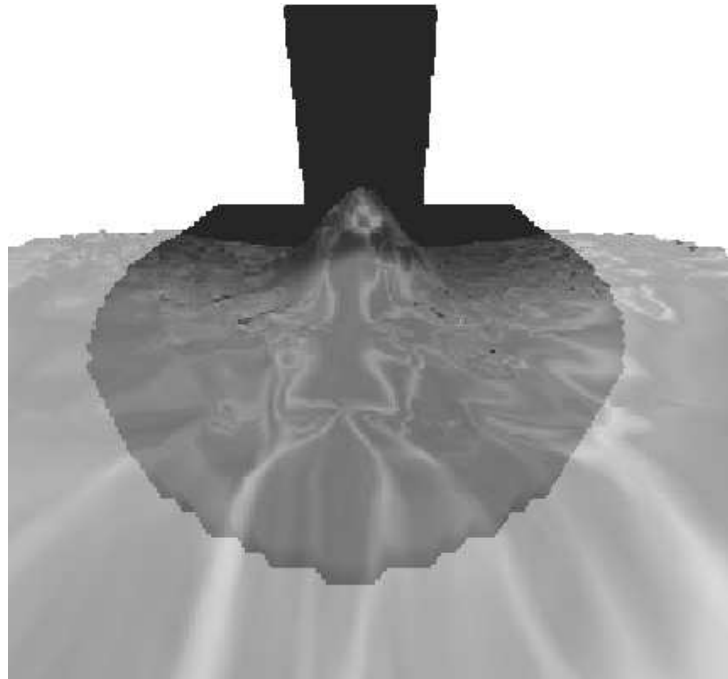


Figure 4.7: *A quantitative visualization of the Titanic; different pressures have been colored differently (VRMLVIEW screenshot).*

4.3 Summary

While developing a CFD visualization method, as part of the pilot project, two presentations of green water simulations have been created. A brief report of both procedures is presented.

The first presentation involves the spatial visualization of the distribution of the pressure. Having stored pressure values all over the surface of the bow of a ship, including a vertical structure on top of its deck, the lowest pressure was assigned a blue and the highest a red color, with intermediate colors in between. The initial result in VR showed the adequacy of the three-dimensional movie for data inspection, since the existence and location of anomalies was clearly visible. Unnatural pressure peaks were visible at different locations, but always near the transition from water to air, the free surface. Besides, stationary water drops in mid air, their size dependent on the computational resolution, were disturbing the scene. For the final edition of the pressure movie, both annoying anomalies have been filtered out manually in COMFLO. While the hovering drops can be largely prevented by increasing the resolution of the computation, avoiding the unwelcome pressure peaks is to be achieved in COMFLO versions currently under development.

At the start of the project a bow was overflowed by waves, and extended from bow to stern to complete the image of a real ship. While this green water simulation had already locally been referred to as ‘Titanic simulation’, finally this reference became true. Towards the end of the project we came across a VRML model of the real Titanic and not much later the realistic model was included in our computations. The bow of the Titanic model (in an adapted representation) was read by COMFLO and provided with waves manipulated in such a way that they overflowed the Titanic model. In addition, a realistic environment surrounds the final result, including the roaring sound of waves in combination with a howling wind.

In chapter 5 the performance of both pilot presentations is discussed.

Chapter 5

Performance

So far we have not really investigated the relationship between the performance requirements of our CFD visualization on the one hand, and the hardware capabilities on the other hand. Before the indication of the existence of potential difficulties with respect to this relationship in our application area, solutions to possible conflicts are suggested.

5.1 Performance Optimization

To reduce the chance of friction in the relationship between requirement and capability, or to solve existing problems, several solutions exist. Options to optimize the relationship that can be integrated in the visualization process (COMFLO \rightarrow VTK \rightarrow VRMLVIEW) are put together here. The first five options, indicated by '◇', can directly affect the frame rate or total number of polygons to be rendered per image, *i.e.* the performance. The other options are only effective in specific advantageous situations.

- ◇ The *frame rate* itself can be manually coded. Or, if too high, VRMLVIEW itself determines the maximum frame rate.
- ◇ Render only *parts* of the data set to be visualized, for example only half of the domain.
- ◇ The number of polygons of which an object is constructed can be reduced by applying a *decimation* algorithm in VTK (see results in figure 5.1).
- ◇ A *display list* feature can be switched on or off using VRMLVIEW (using the option `-nolist`, see for more information appendix B). Such a list is a collection of rendering commands, stored by the application in the rendering device. This collection is later invoked during rendering. It appears that with this option switched on the frame rate increases slightly. But since the total length of the list is limited by the hardware, it is not always possible to put all objects together in lists.
- ◇ *Frustum culling*, *i.e.* showing only items contained within a view volume (a user defined rectangular box), is a feature built in in VRMLVIEW.
- *Texture mapping*, with textures detailed as little as possible, can be applied.
- *Backface culling* can be used, that is checking whether a polygon is facing the camera and omitting the polygons facing away from it.



Figure 5.1: *Decimation of the bow. The original triangulation of the iso-surface consisting of 24292 triangles is shown on the left. The middle figure is a reduction to half of the original number of triangles on the left (12146 triangles). On the right, with 2690 triangles the target reduction of polygons of 90% is only approximated and not completely accomplished due to the prevention of distortions. A further reduction can not be achieved without a violation of the original shape.*

- Also helpful could be a feature creating *triangle strips*, *i.e.* putting together triangles in strips, such that n triangles are described by $n + 2$ (instead of $3n$) points and the total numbers of points is reduced. Although this does not influence the performance, the total storage space can be reduced by this feature.

Other ‘extra’ visualization features may influence the performance in a negative sense and would better be avoided as much as possible. For example:

- *Opacity* influences the performance in a negative sense. A transparent surface would permit the viewer to see polygons on the other side of that particular surface as well, such that a positive effect of backface culling would disappear for that surface.

Because the application of the options is not always straightforward, some remarks have to be made.

Remarks

The most important restriction involving our visualization of CFD applications is the total number of polygons that can be rendered. The first two straightforward options (adjusting the frame rate or splitting up the data set) are not necessarily annoying and can even be convenient when inspecting large data sets. The third and seemingly also pretty straightforward measure, decimation, needs special attention. This decimation option is a kind of filter applied to the simulation data.

- Except for presentations purely meant to be shown to a general public, researchers do not at all like the idea of any filtering routines applied to their output data. The fear of losing relevant information by applying filtering operations not controlled by themselves

is always present. For *any* irregularity could indicate the presence of an error that needs investigation. An example of an unfortunate loss of accuracy, after the application of decimation, is shown in figure 5.2. Too few triangles are left to represent the pressure distribution accurately.

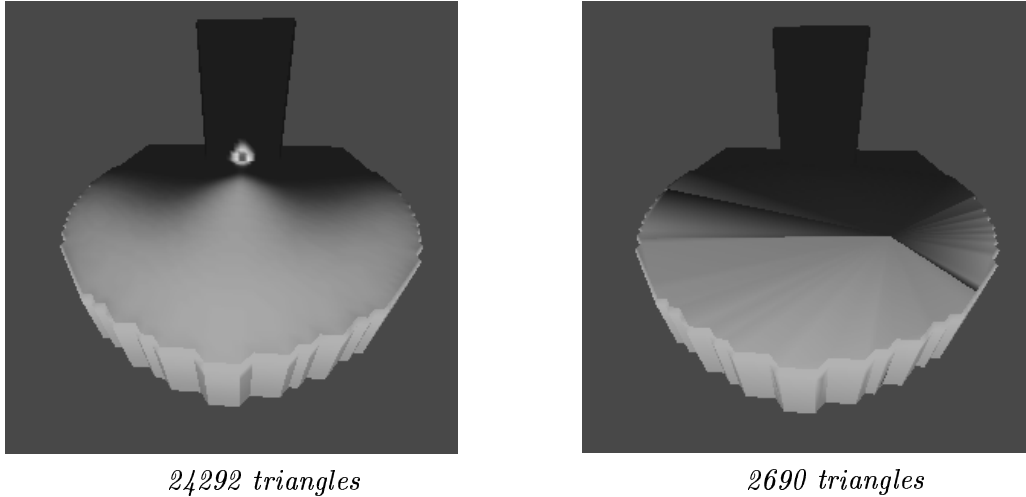
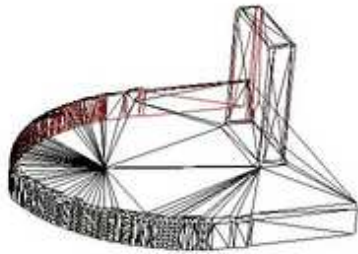


Figure 5.2: *Coloring of the pressure before and after decimation. The left illustration is identical to the one showed in 4.7, having the water surface omitted.*

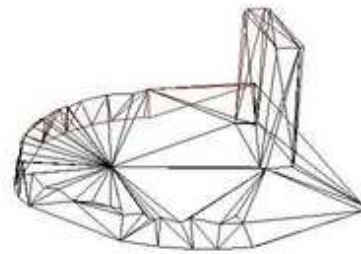
- When the presence of a large number of polygons cannot be reduced and has to be accepted, a way of treating them more efficiently can be applied, such as putting them together in polygon strips. These polygon strips are collections of triangles or quadrangles together forming a strip. In VTK quadrangles have to be triangulated, *i.e.* split up in combinations of triangles, before series of triangles can be put together in strips. The creation of triangle strips by VTK may however interfere with other polygon strips creating features. Our own VRML viewer namely creates standard polygon strips, meanwhile accounting for the rendering order of the polygons. This VRMLVIEW rendering order might differ from the order of triangles in VTK triangle strips, which could matter in the performance. Since we can not test all possible situations, we choose to avoid uncertainties and do not apply the VTK triangle strips (as long as storage space is not a problem).

When satisfied with losing *some* information from the original data set, decimation may still not provide a solution for several possible reasons.

- High reductions by decimation can come with severe errors (see figure 5.3 on the right for an example of topology violation).
- When the preservation of the original topology is a requirement, decimation can sometimes be not at all effective. Sometimes a maximum reduction of no more than 10% of the total number of polygons can be reached, if the original topology to be decimated is complex. When large numbers of polygons are involved however, any reduction is relatively successful in minimizing the storage space.



2690 triangles



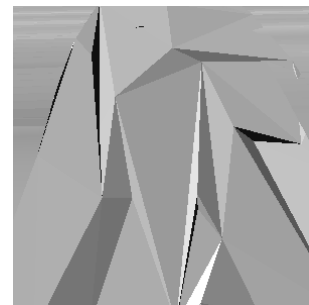
240 triangles

Figure 5.3: *Decimation of the bow. Now the reduction is set to 99%. On the left, distortions of the original shape are prevented. On the right, the target reduction is achieved without explicitly preserving the original topology. Clearly, at the right bottom of the image the original shape is violated in the right figure.*

- When decimating moving surfaces, one can notice strong differences between polygon sizes or locations, when studying two consecutive images. This is illustrated for two images in figure 5.4, an interval of 0.2 s apart from each other. In a movie, the viewer may therefore be bothered by annoying flickering reflections off the surface. In such a case lighting properties of the moving object might need adjustment (see appendix A.2.4).



382.0 s



382.2 s

Figure 5.4: *Decimation of corresponding parts of two consecutive images containing a moving surface, 0.2 s apart. Clearly, triangles do not vary smoothly from image to image, simply because the applied decimation method can not account for that. Setting the surface properties conveniently prevents annoying flickering reflections.*

Summary

Despite the application of all optimization methods suggested, either the performance or the result can still be not satisfactory. Probably dozens of other visualization options do exist, but most of them are useless in improving the performance considerably. If the standard operations mentioned do not offer a solution, a completely different approach might be needed. One LOD (Level Of Detail) kind of improvement suggested by Van Dam *et al.* [4] is to adapt the accuracy of the projection to the capability of the human eye; close to our focal point we can observe more detail than outside this specific region. For example, if we focus on a painting hanging on the wall, we observe other objects in our environment less sharply. Or we can, instead of adjusting the visualization of the data, facilitate the complete process by tackling the basis. We can manipulate CFD data from the source: the CFD program. Reducing the complexity of data sets by CFD researchers seems however quite illogical, with respect to the progressive approach typical to researchers, who are usually interested in increasingly complex problems. Therefore, the main progression is thought to be found in hardware development.

5.2 Requirements and Capabilities

In the research area of CFD, the relationship between presentation requirements and hardware capabilities appears to be one not always free from problems. To show this, the experimentally noticed hardware performance limits are presented. We can be satisfied with these experimentally determined restrictions instead of theoretical ones, thanks to general visualization experiences. For, it appears that an optimal performance can only be reached under ideal circumstances, while such circumstances in practice never, or seldom, occur.

5.2.1 Capabilities

The limitation to the visualizations is created by the hardware capabilities of the ONYX. These capabilities can be demonstrated by the maximum number of polygons that can be generated at a certain frame rate. A minimum frame rate to perceive a smooth transition from one frame to the next is about 10 frames per second (fps). In the Cube this is independent of whether an object is stationary or moving, since there is always relative motion because a user cannot keep his head perfectly still and changes his/her viewing direction all the time. Therefore polygons have to be rendered all the time at different positions, *no matter* whether a 3D image (through which we can navigate in time) or a 4D movie (3D images varying in time) is involved. Now we have determined how the hardware capability can be demonstrated, we are going to investigate it with an example.

In practice, for the model of the real Titanic, constructed of 107,070 polygons provided with textures, the frame rate is influenced by the number of projection screens over which the Titanic is distributed. If all polygons are concentrated on a single screen the maximum frame rate is 8 fps, which can occur when observing from a considerable distance. This is then the lowest maximum frame rate. While observing the Titanic closer, it is distributed over several screens. In that case, the frame rate can increase to a highest maximum number of 24 fps. Apparently the frame rate is restricted by the screen containing the largest number of polygons. Obviously, in case of the Titanic it can not be guaranteed that the visualization is accompanied by a smooth transition, since the frame rate is possibly restricted to a maximum

of 8 fps, while 10 fps is required. Of the system *capability*, it can now be said that within one second a number of 8 times 107,070 (textured) polygons, thus about *800,000 (textured) polygons per second*, can be rendered.

5.2.2 Requirements

Since in the application area of CFD the computational resolutions can vary considerably, we first focus on the current maximum resolution on desktop computers. After that we consider the specific requirements of our Titanic simulations.

General CFD Requirements

We specify the performance requirements of the visualization only with respect to our area of interest, thus the visualization of Computational Fluid Dynamics applications. In CFD applications, a total number of computational cells in three-dimensional desktop computations of the order of $n = 100 \times 100 \times 100 = 1,000,000$ cells is not unusual. The regions of interest consist of iso-surfaces, being two-dimensional subsets of the three-dimensional space. For that reason, the maximum number of relevant cells is reduced by a factor of about the cubic root, to a number of the order $\frac{n}{\sqrt[3]{n}} = n^{2/3} = 10,000$, for unfolded iso-surfaces. Since each rectangular cell face is further triangulated to a combination of two triangles, the estimated number of polygons per iso-surface is thus doubled to about 20,000. Relating this to the desired frame rate of 10 fps, a capacity of 200,000 polygons per second would be necessary to display such a surface smoothly. Since this is not much smaller than the capacity of 800,000 polygons per second, performance difficulties are expected for slightly complex topologies, computations with more computational cells, or several iso-surfaces. For example, severely deformed iso-surfaces can contain a few times the original 20,000 polygons per surface.

Specific Requirements

Titanic model

Additional to the model of the Titanic, in our case we also want to render objects like moving waves. But there is still no need to despair; when wandering around the decks of the ship, not the total amount of polygons is visible in our virtual environment and at the same time the projection of the Titanic is spread over a maximum of 4 screens (three walls and the floor). Both facts affect the frame rate positively.

With our CFD program, COMFLO, a series of waves overflowing the Titanic is calculated. The size of the computational domain used is $150 \times 40 \times 50$ cells. By VTK the water surface is transformed to a collection of polygons. The number of polygons contained in the water surface varies after decimation from about 20,000 for a smooth surface up to 50,000 for a sincerely deformed surface. Then, in combination with the surroundings, the total number of polygons in the complete scene can add up to somewhere around 160,000 polygons. The resulting maximum frame rate that can be achieved is only 5 fps. For this specific example, this frame rate seems to be on the edge of acceptance however.

Note: When putting a series of frames together in a movie, for some reason the collection of files can not exceed the size of 512 MB (while a main memory size of 2 GB is claimed).

However our movie containing the Titanic model, the moving water surface, and other accessories was in total about 300 MB in size.

Colored pressure

The initial movie with a colored pressure tested in the Reality Cube (without applying decimation) showed some hesitation, when due to severe deformations of the water surface the amount of polygons in this surface suddenly increased. In the final version (on a $80 \times 80 \times 40$ grid) however, this performance problem has been avoided. To prevent that hesitation from happening, decimation has been applied and hanging drops have been omitted. Since details on the pressure distribution are the main topic of interest in this application, decimation of the bow is left out of consideration (see also figure 5.2). Decimation of the moving water surface however appears to be sufficient to improve the performance; the number of polygons of which the surface is constructed then varies from 3,000 to 15,000 instead of 20,000 to 50,000. A frame rate of 10 fps is now easily obtained.

Conclusion

When using full modern computational resources, even at a desktop computer, CFD programs can generate more data than can be easily visualized. The frame rate of 10 fps, required for a smooth perception of the transition of consecutive images, can be difficult or impossible to achieve when iso-surfaces, as two-dimensional subspaces of the three-dimensional computational domain, have to be rendered.

Obviously, future visualization hardware improvement provides a solution, but at the same time the development of simulation hardware will result in a growth of the size of data sets. A cruel solution presented is to reduce the number of polygons by applying decimation, but usually such a loss of information is not at all tolerated by researchers.

Chapter 6

Conclusions

In our current world, we have to deal with mountains of data arising from all kinds of sources. And these mountains are still growing. To help us analyze and interpret that amount of data, graphical visualization is thought to offer us a solution.

Especially this kind of visualization is dependent on our visual system, to which more than half of our total amount of neurons is dedicated. Since most of our visual experience is three-dimensional, it is pretty straightforward to make use of our stereoscopic vision by invoking a three-dimensional perception in Virtual Reality. Immersive Virtual Environments like the Cube, in which the user is tracked for his orientation like in reality, can in fact even surpass experiences in non-immersive VEs. Although VR systems still need thorough psychological investigation, results already published are promising. VR appears to be capable of resembling real world situations correctly, since human interaction with this world corresponds to real life behavior. In any case, the main advantage of computer generated virtual environments is that they can always be easily manipulated and controlled. Further, in a stereo VE no distinction is made between three-dimensional images and (4D) movies, since every moment in time the orientation of the user with respect to the object visualized changes, no matter whether the object is moving or the user.

One of the scientific areas producing lots of data and adopting techniques of visualization for proper analyses is the area of Computational Fluid Dynamics. In domains currently containing up to about a million computational cells, the flow of fluids is simulated by the CFD computer program COMFLO. Until the beginning of this research, at the University of Groningen the visualization, also of three-dimensional data, had been restricted to two-dimensional projections to desktop monitor screens. In this thesis a methodology has been developed to expand the area of CFD data analysis to Virtual Reality, in the relatively new Reality Centre of the Centre for High Performance Computing and Visualization. As an appropriate file format to present three-dimensional visualizations, the VRML97 format has been selected. For this format, the HPC/V has an appropriate VRML viewer available. This viewer meets our demands and is capable of showing VRML97 files in the Reality Cube and Theatre, and at a Linux desktop. To take care of the metamorphosis from COMFLO data to VRML files, the Visualization Toolkit VTK is preferred over other visualization systems like Matlab and AVS(/Express).

As a pilot project, the smashing of green water on a ship has served to test and interactively tune our proposed CFD visualization methodology. Besides, the coloring of pressure values on the solid surface of a ship, to which before the project already had been referred to as ‘Titanic simulation’, a model of the real Titanic has been overflowed by green water. Meanwhile all CFD visualization aspects have been met. They consist of data inspection, getting a quantitative spatial impression of the pressure distribution, and presenting a version of which the realism is emphasized specially for a general public. Both resulting movies have been included in the scientific visualization demonstration set of the Reality Centre of the HPC/V.

The major problem encountered during the project is the hardware capacity of the VR systems. The capacity can be demonstrated by the maximum number of polygons that can be rendered per image at a certain frame rate. When rendering the Titanic model in combination with moving waves we can unfortunately only approach and not obtain the desired frame rate of 10 fps to perceive a smooth transition. However, in practice it is in this case not really disturbing the user. The purely scientific pressure visualization poses in any case no performance difficulties yet, when applying one of the performance optimization features presented, namely the decimation option to the moving water surface. But for this visualization the maximum number of computational cells has not been used so far. For the resulting iso-surfaces also performance difficulties are expected.

Prospect

In the near future, probably the first developments related to this project consist of creating other CFD visualizations in VR than green water simulations. Adapting existing scripts to the case specific demands should be sufficient to accomplish such a task. However the most important future development is not controlled by ourselves, since it concerns the evolution of both simulation and visualization hardware. Although CFD researchers will probably always keep the desire for an increasing performance, perhaps (but unlikely) one day visualization scientists will be content for a moment when the quality of the hardware presentation exceeds the perceptual possibilities of our eyes. Apart from the waiting for a ‘better future’, a more actual application that could be developed is computational steering. However for the moment large-scale interactive computational steering, such as visualizing real time simulated wake-vortices caused by aircrafts (Modi [11]), is beyond our possibilities when looking at our simulation time for the Titanic (up to weeks of simulation for minutes of real time) and our visualization performance. One quick step towards this goal can be parallel CFD simulations and is not an unrealistic development.

Appendix A

VTK Related Files

This appendix is meant to provide an overview of VTK features assisting the construction of a CFD visualization. For that purpose, first CFD data have to be made readable by VTK. The requirements imposed on the visualization in section 4.1 are to be achieved by including them in VTK applications written in `tcl`.

Design

The realization of the specific requirements for the visualization of the Titanic is illustrated by presenting a framework of general elements (section A.2.1) in which desired objects (section A.2.2 and A.2.3) can be included. All quantitative aspects coming up in the `tcl`-scripts are put together in section A.2.4. Having exported the result to VRML, it can be shown by the use of VRMLVIEW (appendix B), either in Virtual Reality or at a desktop. But before all, data are to be made readable by VTK.

A.1 Data File

A file format readable by VTK that is easy to write by hand or a computer program (in this case COMFLO) is the standard VTK input file format. It consists of five parts, of which a description with information relevant to our CFD visualizations is shown below. (More information on this file format and other formats can be found in <http://public.kitware.com/VTK/pdf/file-formats.pdf>.)

- The first line is to contain both a file identifier and the version number. It should be exactly `# vtk DataFile Version` followed by a single space and the version number. Concerning the versions, older are compatible with newer.
- Second is the header. It consists of 256 characters maximum, that may contain any information.
- The header is to be followed by the file format, here `ASCII`.
- Next is the dataset structure that describes the geometry and the topology of the dataset. In our case, it concerns a structured point dataset, that can be either one-, two-, or three-dimensional. Dimensions `jmax`, `kmax`, `imax` are greater than or equal to one, while the spacing of the grid in y -, z -, and x -directions respectively should be

greater than 0. These terms refer to the corresponding variables used in COMFLO. Note the difference between the orientation of the coordinate systems of COMFLO and VRML, to which format VTK is to export. The difference is also illustrated in figure A.1.

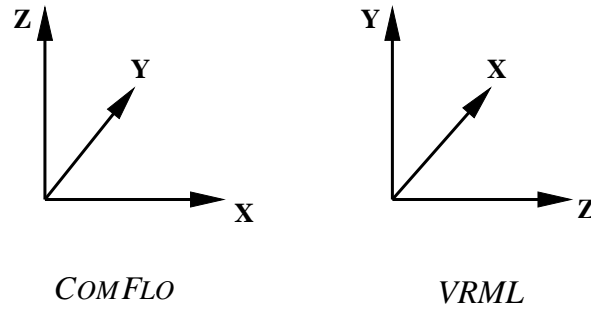


Figure A.1: *The difference between the coordinate systems of COMFLO and VRML*

- Finally the dataset attributes are described. The number of cells is attached to the keyword `POINT_DATA`. Among other types, `SCALARS` can be defined. With this type a `dataName` can be associated used by the VTK data readers to extract data. The `dataType` used in our examples is `float` for floating point representations. An optional `LOOKUP_TABLE` is not specified and set to `default`. Finally, the actual values are included.

An example of a standard VTK input file is shown below. It contains the bow of the Titanic. Except for the obligatory first line, a line starting with “#” contains a comment. The actual values, 374544 in number, are not shown. The structure of these values can be read from the parts of the FORTRAN routine included hereafter.

vtk data file

```
# vtk DataFile Version 2.0
Water surface
ASCII

DATASET STRUCTURED_POINTS
DIMENSIONS      84   39  154
ORIGIN          0.000 0.000 0.000
SPACING         1.061 1.000 1.447

POINT_DATA 504504
SCALARS scalars float
LOOKUP_TABLE default
```

An example of a standard VTK input file. Except for the obligatory first line, a line starting with “#” contains a comment.

The generation of such a VTK input file can be accomplished by the inclusion of a routine in the FORTRAN CFD program similar to the following one. This particular piece is constructed to prepare COMFLO output for a presentation in VRML, with its different coordinate system, as seen in figure A.1.

FORTRAN routine

```
C
      SUBROUTINE VTK3D
C
      IMPLICIT NONE
C
C
      INTEGER IMX, JMX, KMX
      PARAMETER (IMX = 164, JMX = 104, KMX = 54)
C
C      (...)
C
      COMMON /SPCNG/ CELX, CELY, CELZ
      REAL CELX, CELY, CELZ
```

```

C      (...)
C      In the first loop the grid spacing is calculated.
      IF (CYCLE .EQ. 0) THEN
          CELX = (XMAX - XMIN) / MAX
          CELY = (YMAX - YMIN) / JMAX
          CELZ = (ZMAX - ZMIN) / KMAX
C      In this case the spacing is scaled:
          CELMAX = AMIN1(CELX,AMIN1(CELY,CELZ))
          CELX = CELX / CELMAX
          CELY = CELY / CELMAX
          CELZ = CELZ / CELMAX
      ENDIF
C
      OPEN (UNIT=21, FILE=MATDIR(1:MATLNG)//'fs'//NUMBER//'.vtk')
      open (21)
      WRITE(21,'('# vtk DataFile Version 2.0'))')
      WRITE(21,'('Water surface'))')
      WRITE(21,'('ASCII'))')
      WRITE(21,'(''          '''))')
      WRITE(21,'('DATASET STRUCTURED_POINTS'))')
      WRITE(21,'('DIMENSIONS ',3I5))JMAX,KMAX,IMAX
      WRITE(21,'('ORIGIN ', 3F8.3))0,0,0
      WRITE(21,'('SPACING ',3F8.3))CELY,CELZ,CELX
      WRITE(21,'(''          '''))')
      WRITE(21,'('POINT_DATA ',I7))JMAX*KMAX*IMAX
      WRITE(21,'('SCALARS scalars float'))')
      WRITE(21,'('LOOKUP_TABLE default'))')
      WRITE(21,'(3I5)')
      DO I=1, IMAX
          DO K=1, KMAX
              WRITE(21,'(500F5.2)') (FS(I,J,K), J=1, JMAX)
          END DO
      END DO
      CLOSE(21)
C
C      (...)
C
      NRM3D = NRM3D + 1
C
      RETURN
      END
C
C      End of subroutine VTK3D.

```

An example of a FORTRAN routine designed to generate a VTK input file. Positions where FORTRAN statements have been omitted are indicated by “C (...)”.

This routine is called in the main program, as frequently as indicated in the input file `comflo.in`. In this case the parameter `NRM3D`, originally meant to determine the number of Matlab plots, is used. Since in the routine `MATL3D` the same counter `NRM3D` is adopted, the statement `NRM3D = NRM3D + 1` is eliminated from subroutine `MATL3D` if this subroutine is still called.

VTK3D calls within the FORTRAN main program

```

C Before the time cycle, at t=0:
      IF ((DTM3D .LE. TMAX) .AND. (T .EQ. 0.0)) CALL VTK3D
C
C   (...)
C
C In the time cycle:
      IF (T+0.5*DT .GE. DTM3D*FLOAT(NRM3D)) CALL MATL3D
      IF (T+0.5*DT .GE. DTM3D*FLOAT(NRM3D)) CALL VTK3D

```

Main program calls of the FORTRAN routine VTK3D.

A.2 VTK Elements

The process of generating images using computers is called rendering. The VTK applications to render objects are written in the script language `tc1`. A number of seven basic elements that are used regularly to render a scene can be distinguished. That number of basic elements can be split up. There are four general elements that function as a framework in which objects to visualize are included, possibly accompanied by other VTK elements.

- `vtkRenderWindow` creates rendering windows.
- `vtkRenderer` coordinates the process of rendering.
- `vtkLight` provides a light to illuminate the scene.
- `vtkCamera` defines aspects as the viewpoint, the focal point, and the view angle.

The actual object to visualize is referred to as an actor. Within this VTK class, other classes can be included.

- `vtkActor` represents the actual object to be visual. Both information on its properties and its position are contained in the actor.
- `vtkProperty` contains the appearance properties of the surface of actors. These properties include color, transparency, and lighting properties explained in section A.2.1.
- `vtkMapper` describes the geometry of the actor.

The general VTK elements are described in section A.2.1. Specific information on actors is contained in section A.2.2. Additional elements are shown in section A.2.3.

A.2.1 General Visualization Elements

Certain general aspects of visualization always have to be accounted for. Obviously, to see objects a *light source* has to be present. Such a light source can be a local light source with a certain volume located at a finite distance from the scene, but also a point light source at an ‘approximately infinite’ distance. Convenient consequences of an infinite distance between light source and the scene lighted apply for the light intensity and the light emission. In that case, the light intensity is independent of the distance to the light source, thus constant, and the emitted rays of light strike the scene in parallel.

Light rays, having arrived at the object, can be reflected or absorbed. Direct reflections off a shiny object are indicated as *specular light*. In figure A.2 a series of spheres with an increasing amount of specular light is shown. Furthermore, the amount of light reflected is dependent on the *specular power* or *shininess* of the object. An increase of this object property indicates a higher decrease of specular reflection as the angle of incidence deviates more from a perfect reflection. This effect of an increase of specular power is illustrated in figure A.3. Caused by the direct reflection of light off objects is an indirect lighting, referred to as *ambient light*.

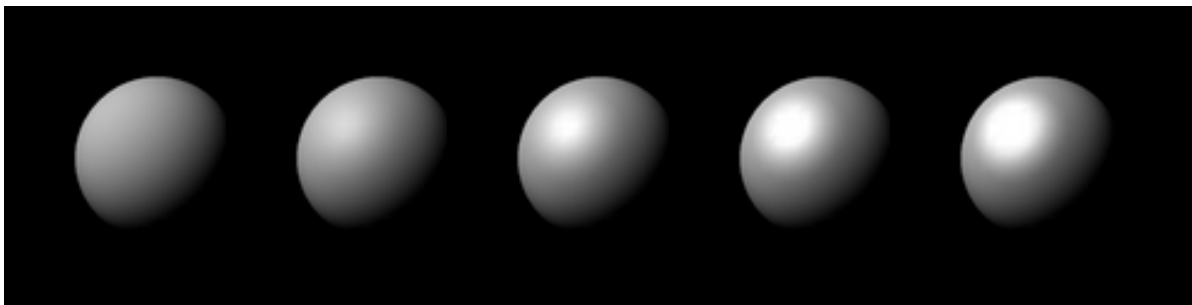


Figure A.2: An increase of specular light (0, 0.25, 0.5, 0.75, and 1.0, specular power 20) accompanied by an increasing amount of reflection (produced using VTK).

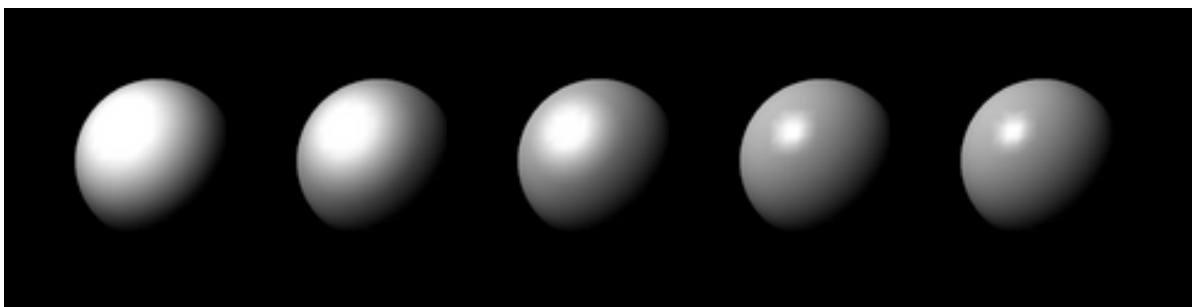


Figure A.3: An increase of specular power (5, 10, 20, 80, and 128, specular light 0.75). Clearly the amount of specular reflection decreases as the angle of incidence deviates more from a perfect angle while the specular power increases (produced using VTK).

The lighted scene is observed from a viewpoint, initially defined by `vtkCamera`. A `tcl`-script containing the rendering window and both a light and a camera is shown below. For the

visualization of the Titanic, this particular script serves as a framework in which desired objects represented by `vtkActors` (section A.2.2) and other VTK features (section A.2.3) can be included.

Notes:

In the VTK scripts included in this thesis the following lay out is introduced, illustrated with examples from the VTK script below.

- A “#” followed by at least two spaces indicates a comment. Example:
`# define a renderer, to be referred to as 'ren1'`
- Starting from the first column, a visualization element is initialized. Example:
`vtkRenderer ren1`
- Two spaces followed by a statement, thus starting from the third column, an already initialized visualization element is changed or is assigned a certain property. An example involving the previously initialized renderer `ren1`:
`ren1 SetAmbient 0.1 0.1 0.1`
- Other aspects may also start from the first column:
`catch {load vtkc1}`

Whereas this spacing is introduced in this thesis to serve an orderly presentation, in practice the spacing in the `tcl`-scripts does not influence any aspects when initializing or changing visualization elements.

Concerning their directories, references to VTK data files (with extension `vtk`, all similar to the one presented in section A.1) or textures (here in `bmp` format, some other file formats are also supported) can vary. Since one simulation was more appropriate for the pressure visualization and the other includes the model of the real Titanic, data files can originate from different computations in different directories. Textures can be stored in separate directories.

framework

```
catch {load vktcl}
source ~/.wishwi
source vtkInt.tcl
source colors.tcl

# define a renderer, to be referred to as 'ren1'
vtkRenderer ren1
# adjust the ambient light color in the VRML format, results in
# an adjustment of ambientIntensity (VRML)
ren1 SetAmbient 0.1 0.1 0.1
# although finally the background is changed in the movie,
# it is conveniently set to white (= RGB 1 1 1) for instantly
# viewing the result
ren1 SetBackground 1 1 1

# define a renderwindow, to be referred to as 'renWin'
vtkRenderWindow renWin
# add the renderer 'ren1' to this renderwindow 'renWin'
renWin AddRenderer ren1
# an interactor to serve mouse/keyboard control
vtkRenderWindowInteractor iren
iren SetRenderWindow renWin

# define a vtkLight, be referred to as 'light'
vtkLight light
# define a light source at an infinite distance:
# equal light intensity + parallel lighting
light SetLightTypeToSceneLight
# to adjust the directional light in VRML
light SetPosition -150 150 150

# add the light to the renderer 'ren1'
ren1 AddLight light
```

```

# define a camera 'aCamera' by defining the active camera to be
#   'aCamera'
set aCamera [ren1 GetActiveCamera]
# define the viewpoint of the camera, convenient for the reset option
#   of the VRML viewer
[ren1 GetActiveCamera] SetPosition 20 55 220
# define the view angle
[ren1 GetActiveCamera] SetViewAngle 90
# define the focal point for the orientation
[ren1 GetActiveCamera] SetFocalPoint 20 0 50

# set renderwindow size
renWin SetSize 500 500

# render the image
iren Initialize

```

A framework of the general VTK elements `vtkRenderer`, `vtkRender`, `vtkLight`, and `vtkCamera`, in which the actors (section A.2.2) and other VTK features (section A.2.3) can be included. A line starting with “#” contains a comment.

A.2.2 Actors

A scene usually consists of a light, a camera, and actors. An arbitrary number of actors can be attached to the framework of section A.2.1. Primary objects to be included as actors are for the visualization of the Titanic the bow of the ship itself and the wave overflowing it. To contribute to the sense of realism, a carpet of waves surrounding the ship is also included in the framework. Comments are indicated with “#”.

Note: In the scripts some of the elements that have been shown in earlier scripts are not repeated. In other cases elements are repeated, but not the earlier shown comments.

The Titanic

Before the real life VRML model of the Titanic (before it had sunk) had been ‘discovered’, we referred to the bow used in our COMFLO simulations as the ‘Titanic’. From the file `fb0000.vtk` its geometry is read. This file is similar to the one presented in section A.1, containing information on the presence or absence of solid material in every computational cell. With this information an isosurface representing the solid surface of the bow is created by VTK. Next, a texture is attached to the bow by a `vtkTextureMap`.

'Titanic' bow

```
# define surface properties to be assigned to the Titanic
vtkProperty solidproperty
  solidproperty SetAmbient 0.2
  solidproperty SetSpecular 1
  solidproperty SetOpacity 1
# SpecularPower (VTK) / 128 = shininess (VRML)      (25.6/128=0.2)
  solidproperty SetSpecularPower 25.6

# read the data that define the Titanic: a value 1 indicates the
#   presence of the ship, a value 0 the absence, such that
#   an isosurface through the value 0.5 shows the Titanic
vtkStructuredPointsReader fbreader
  fbreader SetFileName "../../titanicwavecomflow/data/fb0000.vtk"
  fbreader Update

# there is some superfluous information contained in the data file
#   that is to be cut off; define the region of interest
set dims [[fbreader GetOutput] GetDimensions]

set dim1 [lindex $dims 0]
set dim2 [lindex $dims 1]
set dim3 [lindex $dims 2]

set xmin 3
set xext [expr $dim1 -3]
set ymin 3
set yext [expr $dim2 -3]
set zmin 3
set zext [expr $dim3 -3]

# select the region of interest (Volume Of Interest)
#   from the dataset read in 'fbreader'
vtkExtractVOI fbextract
  fbextract SetInput [fbreader GetOutput]
  fbextract SetVOI $xmin $xext $ymin $yext $zmin $zext

# the value of the isosurface is 0.5
set fblevel 0.5
vtkMarchingCubes fbiso
  fbiso SetInput [fbextract GetOutput]
  fbiso SetValue 0 $fblevel
```

```

# scale the texture coordinate to get some repeat patterns, if desired
vtkTransformTextureCoords xform
  xform SetInput [fbiso GetOutput]
  xform SetScale 1 1 1
# in fact SetScale with parameters 1 1 1 does not transform

# generate texture coordinates
vtkTextureMapToSphere fbisoMapper
  fbisoMapper SetInput [xform GetOutput]

# vtkDataSetMapper internally uses a vtkGeometryFilter to extract the
# surface from the triangulation. The output (which is vtkPolyData) is
# then passed to an internal vtkPolyDataMapper which does the
# rendering.
vtkDataSetMapper mapper
  mapper SetInput [fbisoMapper GetOutput]

# A texture is loaded using an image reader. Textures are simply images.
# The texture is eventually associated with an actor.
vtkBMPReader bmpReader
  bmpReader SetFileName "titanicpwave/textures/metal26.bmp"

vtkTexture atext
  atext SetInput [bmpReader GetOutput]
  atext InterpolateOn

# define the actor and include its properties
vtkActor fbisoActor
  fbisoActor SetMapper mapper
  fbisoActor SetTexture atext
  fbisoActor SetProperty solidproperty
  [fbisoActor GetProperty] SetSpecularColor 1 1 1

# assign our actor to the renderer
ren1 AddActor fbisoActor

```

The bow of the 'Titanic'.

To optimize the performance, a reduction of the total number of polygons is useful. The decimation method `vtkDecimatePro` in VTK is appropriate for this purpose. With this method the total number of triangles can be reduced with a target reduction `SetTargetReduction`. Polygons that are not triangles can be triangulated before this operation using VTK's `TriangleFilter`. Distortions by the reduction can be prevented by prohibiting errors to a certain level (`SetMaximumError`), meanwhile obviously imposing a limit to the target number.

After triangulation, another improvement in favor of saving storage space can be found in the application of `vtkStripper`. This filter puts together triangles in strips, such that n triangles are described by $n + 2$ (instead of $3n$) points, that is each set of three points $(i, i + 1, i + 2)$

with integer $i \in [0, n]$ defines a triangle. Although this option seems very convenient, it may interfere with the standard polygon stripping active in VRMLVIEW (see the manual in section B.3). Namely, the stripping by VRMLVIEW in preparation of an optimal rendering of polygons may differ from the standard VTK triangle stripping. For that reason omitting the `vtkStripper` may be preferable.

In the example script below, a decimation of the bow of the Titanic is shown. After decimation, also `vtkStripper` is applied. The result is shown in figure 5.1, section 5.

decimation

```

vtkStructuredPointsReader fbreader
  fbreader SetFileName "../../titanicwavecomflow/data/fb0000.vtk"
  fbreader Update
    set dims [[fbreader GetOutput] GetDimensions]

    set dim1 [lindex $dims 0]
    set dim2 [lindex $dims 1]
    set dim3 [lindex $dims 2]

    set xmin 3
    set xext [expr $dim1 -3]
    set ymin 3
    set yext [expr $dim2 -3]
    set zmin 3
    set zext [expr $dim3 -3]
vtkExtractVOI fbextract
  fbextract SetInput [fbreader GetOutput]
  fbextract SetVOI $xmin $xext $ymin $yext $zmin $zext

set fblevel 0.5
vtkMarchingCubes fbiso
  fbiso SetInput [fbextract GetOutput]
  fbiso SetValue 0 $fblevel
vtkTransformTextureCoords xform
  xform SetInput [fbiso GetOutput]
  xform SetScale 1 1 1
vtkTextureMapToSphere fbisoMapper
  fbisoMapper SetInput [xform GetOutput]
vtkBMPReader bmpReader
  bmpReader SetFileName "titanicpwave/textures/metal26.bmp"

vtkTexture atext
  atext SetInput [bmpReader GetOutput]
  atext InterpolateOn

```

```

# apply decimation, i.e. specify the desired reduction of the total
#       number of polygons (TargetReduction 0.9 = reduction of 90%)
vtkDecimatePro deci
  deci SetInput [fbisoMapper GetOutput]
  deci SetTargetReduction 0.9
# prevent distortions and set the maximum error to 0
  deci SetMaximumError 0
  deci PreserveTopologyOn

# create as many triangle strips as possible, such that for each
#       strip n triangles can be represented by n+2 points
vtkStripper decistrip
  decistrip SetInput [deci GetOutput]
  decistrip Update

vtkDataSetMapper mapper
  mapper SetInput [decistrip GetOutput]

vtkActor fbisoActor
  fbisoActor SetMapper mapper
  fbisoActor SetTexture atext
  fbisoActor SetProperty solidproperty
  [fbisoActor GetProperty] SetSpecularColor 1 1 1

ren1 AddActor fbisoActor

```

Decimation of the bow of the Titanic.

Besides a real life visualization of the Titanic, an impression of the spatial distribution of the pressure is also part of our interest. Therefore an iso-surface on which this physical quantity has to be colored is indicated. Also minimum and maximum values of the pressure are determined in order to construct a color scale. The part of the script shown is contained in the loop that is introduced on page 57.

coloring

```
# read in the data that are to be colored quantitatively;
#   in pallover****.vtk computed by ComFlo the complete
#   pressure field is contained
vtkStructuredPointsReader color_on_iso
  color_on_iso SetFileName "../..gw/data/pallover$s.vtk"
  color_on_iso Update

# read the data that contain the surface that has to be colored
vtkStructuredPointsReader isosurface
  isosurface SetFileName "../..gw/data/fb0000.vtk"
  isosurface Update

# select the relevant part of the data
  set dims [[isosurface GetOutput] GetDimensions]

  set dim1 [lindex $dims 0]
  set dim2 [lindex $dims 1]
  set dim3 [lindex $dims 2]

  set xmin 3
  set xext [expr $dim1 -3]
  set ymin 3
  set yext [expr $dim2 -3]
  set zmin 3
  set zext [expr $dim3 -3]

# select the relevant Volume Of Interest
vtkExtractVOI isosurfaceextract
  isosurfaceextract SetInput [isosurface GetOutput]
  isosurfaceextract SetVOI $xmin $xext $ymin $yext $zmin $zext

set level 0.5
vtkContourFilter iso
  iso SetInput [isosurfaceextract GetOutput]
  iso SetValue 0 $level

vtkProbeFilter probe2
  probe2 SetInput [iso GetOutput]
  probe2 SetSource [color_on_iso GetOutput]

vtkCastToConcrete cast2
  cast2 SetInput [probe2 GetOutput]
```

```

vtkPolyDataNormals normals
  normals SetMaxRecursionDepth 100
  normals SetInput [cast2 GetPolyDataOutput]
  normals SetFeatureAngle 45

vtkLookupTable clut
# blue..red
  clut SetHueRange 0.67 0
  clut Build

# set the minimum value of the pressure to the atmospheric
#   pressure 1(*10^5 Pa)
  set minv 1
# a maximum of about 75% of the real maximum value (200) results in a
# nice red coloring during several time steps; the real maximum
#   value would show only a red color in one file of the series
  set maxv 150

vtkPolyDataMapper isoMapper
  isoMapper SetInput [normals GetOutput]
  eval isoMapper SetScalarRange $minv $maxv
# not really necessary in this case, apparently the coloring
#   in VTK is default blue..red
  isoMapper SetLookupTable clut

vtkActor isoActor
  isoActor SetMapper isoMapper

# Add the actors to the renderer, set the background and size
  ren1 AddActor isoActor
  renWin SetSize 500 500
# the background is colored sort of blue, to direct your attention
#   to the colored pressure (blue attracts the least attention)
  ren1 SetBackground 0.2824 0.2392 0.5451

```

Pressure coloring on the Titanic

Wave

The iso-surface containing information on the water surface of the wave is constructed largely similar to the iso-surface that resembles the bow of the ship. Main difference is of course the different input file with information on the position of the free surface in every computational cell. Besides, the `vtkProperty` of the actor representing the wave might be changed. When visualizing the pressure on the ship in combination with the water surface, it is for example necessary to make the water surface transparent. Further, the wave is not assigned a texture by VTK. Later, in appendix B it is shown how to assign an optional VRML `ReflectionMap` to the water surface to imitate its reflective property.

Surrounding Wave Carpet

To contribute to the sense of being in mid-ocean, a large carpet of waves surrounding the ship is created.

wave carpet

```
# create a 100x100 plane
vtkPlaneSource plane
  plane SetXResolution 100
  plane SetYResolution 100

# transform the plane by a factor of 1000 on X and Y
vtkTransform transform
  transform Scale 1000 1000 1

vtkTransformPolyDataFilter transF
  transF SetInput [plane GetOutput]
  transF SetTransform transform

# Conveniently editing the VTK Example /Examples/Modelling/Tcl/expCos.tcl
#   resulted in a carpet of waves. Comments from that example
#   are copied.
# Compute Bessel function and derivatives. We'll use a programmable filter
#   for this. Note the unusual SetInput and GetOutput methods.
vtkProgrammableFilter besself
  besself SetInput [transF GetOutput]
  besself SetExecuteMethod bessel

# The SetExecuteMethod takes a Tcl proc as an argument
#   In here is where all the processing is done.
proc bessel {} {
  set input [besself GetPolyDataInput]
  set numPts [$input GetNumberOfPoints]
  vtkPoints newPts
  vtkFloatArray derivs

  for {set i 0} {$i < $numPts} {incr i} {
set x [$input GetPoint $i]
set x0 [lindex $x 0]
set x1 [lindex $x 1]
```

```

set r [expr sqrt($x0*$x0 + $x1*$x1)]
set x2 [expr 2 * sin(100*$r)]
# 'deriv' might influence the color
#     set deriv [expr -120.0*cos(20.0*$x0)]
set deriv [expr +0.7]

# translate the wave carpet 34m up
newPts InsertPoint $i $x0 [expr 14 + $x2] $x1
eval derivs InsertValue $i $deriv
    }

    [besself GetPolyDataOutput] CopyStructure $input
    [besself GetPolyDataOutput] SetPoints newPts

    newPts Delete; #reference counting - it's ok
    derivs Delete
}

#
# We warp the plane based on the scalar values calculated above
#
vtkWarpScalar warp
    warp SetInput [besself GetPolyDataOutput]
    warp XYPlaneOn
    warp SetScaleFactor 0.5

#
# We create a mapper and actor as usual.
vtkTextureMapToPlane warpMapper
    warpMapper SetInput [warp GetOutput]

vtkTransformTextureCoords zform
    zform SetInput [warpMapper GetOutput]

# vtkDataSetMapper internally uses a vtkGeometryFilter to extract the
# surface from the triangulation. The output (which is vtkPolyData) is
# then passed to an internal vtkPolyDataMapper which does the
# rendering.
vtkDataSetMapper carpetmapper
    carpetmapper SetInput [zform GetOutput]

vtkProperty liquidproperty
    liquidproperty SetAmbient 0.2
    liquidproperty SetSpecular 1
    liquidproperty SetSpecularPower 25.6
# opacity default 1

```

```

vtkActor carpet
  carpet SetMapper carpetmapper
  carpet SetProperty liquidproperty
  [carpet GetProperty] SetSpecularColor 1 1 1

ren1 AddActor carpet

```

Wave carpet.

A.2.3 Other VTK Features

Some features applied that are not explicitly visible can for that reason not be classified as actors. These are presented in this section.

Exporter Scripts

Usually it is desirable to export a visualization to a specific file format. Among the many formats supported by VTK are JPEG and VRML. Occasionally a two-dimensional projection of a scene to the JPEG format is useful in reports. If a presentation is to be shown in Virtual Reality, VRML is a suitable format.

In the script containing the `vtkJPEGWriter` an extra rendering window covers both a 'Save as .jpg' and an 'Exit' option. This extra window is produced to give the opportunity to first change the orientation of the object before exporting. Since in the three-dimensional VRML format changing the orientation is still possible, the `vtkVRMLExporter` is not provided with such an extra window.

`vtkJPEGWriter`

```

# create a frame 'f' containing a save-button and an exit-button
frame .f
button .f.jpg -text "save image as .jpg "
button .f.s -text "exit"
pack .f.jpg .f.s
pack .f

renWin Render
vtkWindowToImageFilter w3if
w3if SetInput renWin

```

```

bind .f.jpg <Button-1> {
vtkJPEGWriter rt_jpegw_dashboard
    rt_jpegw_dashboard SetFileName scene.jpg
    rt_jpegw_dashboard SetInput [w3if GetOutput]
    rt_jpegw_dashboard SetQuality 85
    rt_jpegw_dashboard Write
    puts "scene written to scene.jpg"
}
bind .f.s <Button-1> {
    exit
}

```

A script to save a visualization to a JPEG file. In a window separate from the one containing the visualization, a button is rendered that offers the possibility to first change the viewpoint and/or zoom before saving.

vtkVRMLExporter

```

# VRMLExporter
# all elements included in the renderwindow renWin are exported
vtkVRMLExporter wrlexp
    wrlexp SetRenderWindow renWin
# set the file name to write to, in this case scene.wrl
    wrlexp SetFileName scene.wrl
    wrlexp SetSpeed 4
    wrlexp Write
# to keep an eye on the progression of VTK,
# we write a line to the screen
    puts "scene.wrl written"

```

Export the current actors to VRML.

Loop Script

To cover a (time) series of data files, it is useful to apply a loop on the visualization. An example is shown below, containing a way to read and write data. To prevent an assignment of different characteristics to the same actor, at the end of the loop elements are deleted before entering the next cycle.

loop

```
# open a time loop
for { set k 1700 } { $k < 2001 } { incr k } {

# assign the counter to a string 's'
  if { $k < 10 } { set s "000$k" }
  if { $k > 9 } { if { $k < 100 } { set s "00$k" } }
  if { $k > 99 } { if { $k < 1000 } { set s "0$k" } }
  if { $k > 999 } { if { $k < 10000 } { set s "$k" } }

# to be informed of the progress
  puts "file number $s"

# example data reader
vtkStructuredPointsReader fsreader
  fsreader SetFileName "../../titanicwavecomflow/data/tempwave/fs$s.vtk"

# to save time while producing the series of wrl-files, prevent
# the window from showing up: put a '#' in front of renWin Render
# renWin Render

# example exporter
vtkVRMLExporter wrlexp
  wrlexp SetRenderWindow renWin
  wrlexp SetFileName titanicdata/titanicpwaveserie$s.wrl
  puts "titanicdata/titanicpwaveserie$s.wrl gemaakt"
  wrlexp SetSpeed 4
  wrlexp Write

# delete all elements before starting the next time step
  fsreader Delete
  wrlexp Delete
# etc (delete also all other elements than fsreader and wrlexp)

# close the loop
}
```

A script to generate a series of wrl-files. Examples of reading and exporting data are included.

A.2.4 Summary of Quantitative Aspects

Some translations from VTK to VRML possess quantitative aspects. Table A.1 is meant to give an overview of all quantitative translations from VTK to VRML of interest. The elements of the `tc1`-script are shown on the left with the corresponding VRML elements on the right. Below the table, the application of the separate elements is explained.

VTK		VRML	
<i>class</i>	<i>characteristic</i>	<i>node</i>	<i>field/event</i>
<code>vtkRenderer</code>	<code>SetAmbient</code> α α α ($\alpha \in [0, 1]$)	<code>DirectionalLight</code>	<code>ambientIntensity</code> α
<code>vtkLight</code>	<code>SetPosition</code> y z x	<code>DirectionalLight</code>	<code>direction</code> $-y$ $-z$ $-x$
<code>ActiveCamera</code>	<code>SetPosition</code> y z x <code>SetViewAngle</code> θ (degrees) <code>SetFocalPoint</code> y z x	<code>ViewPoint</code>	<code>position</code> y z x <code>fieldOfView</code> $\frac{\theta}{180} 2\pi$ (radials) <code>orientation</code>
<code>vtkProperty</code>	<code>SetAmbient</code> $\alpha \in [0, 1]$ <code>SetSpecular</code> $\alpha \in [0, 1]$ <code>SetOpacity</code> $\alpha \in [0, 1]$ <code>SetSpecularPower</code> α	<code>Material</code>	<code>ambientIntensity</code> α <code>specularColor</code> α <code>transparency</code> $1 - \alpha$ <code>shininess</code> $\frac{\alpha}{128}$
<code>vtkVRMLExporter</code>	<code>SetSpeed</code> α	<code>NavigationInfo</code>	<code>speed</code> α

Table A.1: *Quantitative VTK characteristics and the corresponding fields or events in VRML, to which automatically is exported by `vtkVRMLExporter`.*

- The `SetAmbient` option within the `vtkRenderer` influences the lighting of all objects present in this renderer.
- This can be overruled by assigning another parameter value to an object property in `vtkProperty`. If a decimation is applied on a moving surface, it is even demanded that the value `SetAmbient` of this moving object is put to 0 to prevent annoying reflections. Other `vtkProperty` changes are straightforward.
- The position of the `vtkLight` with respect to the origin of the scene determines the direction of the parallel light rays of a `DirectionalLight`.
- De `SetPosition` option of the `ActiveCamera` influences the `ViewPoints` becoming available in VRML after exporting the scene. The orientation and the field of view are subsequently established by `SetFocalPoint` and `SetViewAngle`, respectively.
- With the `SetSpeed` characteristic the maximum navigation speed in the Reality Cube is set. This should be in proportion with the size of your objects, keeping in mind that for instance a navigation of 50 m/s past a 10 m long object does not give the passer-by a clear vision of this object. When controlling the navigation from the keyboard at your desktop or in the Reality Theatre, this characteristic is of even greater interest. Then the minimum speed is namely set by this parameter. For example: pressing the “**W**” button twice to move forward causes a velocity twice the `SetSpeed` value.

Appendix B

VRMLVIEW

In this appendix information is contained on VRMLVIEW. The current release is available in *~hess/vrmlview/release* at *hpv.service.rug.nl* (via telnet, for the moment only with access and user permission). It is accompanied by a README file with instructions for installation. Possibly, small changes to the series of `wr1`-files are desirable. These changes can be applied by using the c-shell script below. Next, the one `wr1`-file containing all separate visualization elements can be called with `vrmlview`. The manual for VRMLVIEW (version January 31, 2003) is duplicated in section B.3.

B.1 Optional c-shell Script

In this c-shell script, to imitate the typical light reflections of water a map reflecting a JPEG image is added to both the moving water surface and the carpet of waves surrounding it. Although not really necessary, this reflection map is useful since the reflections at the water surface improve the depth perception. Also the type of navigation is restricted to `FLY`, while the less useful type `EXAMINE` can be omitted. The crease angle results in a smooth shading across the polygonal faces. Changes are saved to the subdirectory `refl`.

Remember that this script is not at all obligatory for the visualization in Virtual Reality or at your desktop. Not any debug option of the VRML viewer is in any case needed, with or without application of this script. Note that an option `-angle` is also available to account for crease angles when using `VRMLview` (see the manual in section B.3). However, its application is not always desirable. For example only some objects, like water surfaces, need smooth shading if the emphasis is on a representation of reality, while for other objects, like ships, the sharp edges in their geometry should be preserved.

optional c-shell script

```
#!/bin/csh -f
mkdir refl
foreach f ( titanicwaveserie*.wrl )
  echo $f
  sed 's/Appearance {/Appearance { texture ReflectionMap
      { url "water.jpg" }/' $f > ! temp$f
  sed 's/"EXAMINE","FLY"/"FLY"/' temp$f > ! temp2$f
  rm temp$f
  sed 's/geometry IndexedFaceSet {/geometry IndexedFaceSet
      { creaseAngle 1.5/' temp2$f > refl/$f
  rm temp2$f
end
sed 's/Appearance {/Appearance { texture ReflectionMap
    { url "water.jpg" }/' titanicwavecarpet.wrl
    > titanicwavecarpettemp.wrl
sed 's/geometry IndexedFaceSet {/geometry IndexedFaceSet
    { creaseAngle 1.5/' titanicwavecarpettemp.wrl
    > titanicwavecarpettemp2.wrl
rm titanicwavecarpettemp.wrl
sed 's/"EXAMINE","FLY"/"FLY"/' titanicwavecarpettemp2.wrl
    > titanicwavecarpettemp3.wrl
mv titanicwavecarpettemp3.wrl refl/titanicwavecarpet.wrl
rm titanicwavecarpettemp2.wrl
sed 's/"EXAMINE","FLY"/"FLY"/' titanicship.wrl > ! refl/titanicship.wrl
foreach f ( titaniciso*.wrl )
  echo $f
  sed 's/"EXAMINE","FLY"/"FLY"/' $f > ! refl/$f
end
```

An optional c-shell script to add reflection maps and change the navigation type.

B.2 Movie Script

The `wrl`-file below containing the movie to visualize has been generated using a FORTRAN file in which just the start (1800) and end (1880) frame, and the cycle interval have to be indicated. VRMLVIEW is to call this particular file, in which all separate files are included. Sound fragments containing the roaring sound of ocean waves in combination with a howling wind are also included. The fragments were manually edited to match the arrival of the waves at the bow, instead of directly relating pressure values (accompanied by some unnatural pressure peaks) to the sound volume. The fragments concerned have the following properties: `wav-format`, a frequency of 44100 Hz, 16 bits, mono. More information on VRML can be found at www.web3d.org/technicalinfo/specifications/vrml97/vrml97specification.pdf.

VRML movie script

```
#VRML V2.0 utf8

# Set the background.
Background {
  backUrl    "back.jpg"
  frontUrl   "front.jpg"
  leftUrl    "left.jpg"
  rightUrl   "right.jpg"
  topUrl     "reflectsky.jpg"
  bottomUrl  "bottom.jpg"
}

# Show objects that do not move during the movie.
Inline { url "titanicship.wrl" }
Inline { url "titanicwavecarpet.wrl" }

# Read the following files one by one.
DEF switch Switch {
  whichChoice 0
  choice [
  Inline { url "titanicpwaveserie1800.wrl" }
  Inline { url "titanicpwaveserie1801.wrl" }
  Inline { url "titanicpwaveserie1802.wrl" }
  # etc
  Inline { url "titanicpwaveserie1878.wrl" }
  Inline { url "titanicpwaveserie1879.wrl" }
  Inline { url "titanicpwaveserie1880.wrl" }
  ]
}

# Define the time interval between two frames.
DEF clock TimeSensor { loop TRUE cycleInterval 0.10000001 }

# Include the combination of two audio fragments: the roaring water
# and a howling wind
Sound {
  source AudioClip {
  startTime 0
  loop TRUE
  url "../Sound/titanicocean16.wav"
  }
  maxFront 100
  maxBack 100
  spatialize FALSE
}
```

```

Sound {
  source AudioClip {
    startTime 0
    loop TRUE
    url "../Sound/titanicwind16.wav"
  }
  maxFront 100
  maxBack 100
  spatialize FALSE
}

DEF script Script {
  eventIn SFTime cycletime
  field SFInt32 state 0
  eventOut SFInt32 frame
  url "vrmlscript:
    function cycletime() {
      state = state + 1;
# We want the movie to start over again after our number of 81 frames
#   have been shown.
      if (state > 81) state = 0;
      frame = state;
    }
"
}

ROUTE clock.cycleTime TO script.cycletime
ROUTE script.frame TO switch.set_whichChoice

```

The url-file containing all separate url-files to be shown in the movie. Except for the first line, “#” is followed by a comment.

B.3 VRMLVIEW Manual

Below, the manual that can also be found at <http://oldwww.rug.nl/hpc/people/guests/vrmlviewer.htm> is shown. Most relevant to the occasional user is the user interface that describes the keyboard and wand control, also illustrated in section 4.2.4. The command line options may be convenient for debugging, although they are not needed for the visualization of the Titanic.

manual

VRMLVIEW

Vrmlview is a stand-alone viewer for VRML97 (www.web3d.org/technicalinfo/specifications/eai/)

index.html). It supports multiple screens using multi-threading or multi-processing, sound and scripting. Most VMRL97 node types are supported (see below). Special features are correct treatment of transparency and reflection mapping across multiple screens. It has an efficient rendering engine based on OpenGL using frustum culling, polygon strips, GL interleaved arrays and display lists. Positional audio is supported through the OpenAL (www.openal.com) library which seems to have some problems (see below).

Libraries

Required graphics libraries:

- GL
- GLU

and one of the following:

- GLUT (single screen)
- MPK Multipipe SDK (single and multi screen)
- CAVELib (multi screen with head tracking)

Optional audio library:

- OpenAL (problems under Linux, no stereo effects, and IRIX, stereo wav file are interpreted as mono and thus played at half the frequency with some noise)

Optional JavaScript library:

- SpiderMonkey (does not compile under IRIX; download mozilla which comes with a compiled libmozjs.so)

Command line options

The command line option `-h` or `-help` displays the usage of `vrmlview`.

Most users will only need three of the command line options:

- `-nq` which controls the smoothness of Cones, Cylinders and Spheres. The default value is 3. When a scene contains many of these objects and the framerate is too low try `-nq 2` or `-nq 1`. This can increase the framerate by up to a factor of 2 or 9 respectively.
- `-smooth` which is necessary to make some scenes look smooth.
- `-angle` which is necessary for vrmf files written by 3D Studio Max, which does not write a `creaseAngle` field in `IndexedFaceSet`'s. This makes objects look flats shaded. To fix this try `-angle 1.5`. Note that the angle is given in radians.

The command line options are (might be out of date): Usage: `./vrmlview [command-line options] <vrml file>`

Options that influence the appearance of the scene:

- smooth** smooth normals over polygons with identical coordinates without identical coordinate indices
- angle** crease angle (rad) for objects without explicit creaseAngle (try 1.5 for 3dsmax output) (default 0.0)
- light** directional lights illuminate the whole scene (3dsmax)
- ambient** minimum ambientIntensity in Material (default 0.0)
- ccw** set ccw to TRUE for all IndexedFaceSets

Options that influence the timing:

- wait** wait for 'Enter' for starting the time
- stop** stop time (default 0.0)

Options that influence the image and audio quality:

- nq** number of quads along 90 degrees of a sphere and cylinder and cone and twice the number of quads along the height of a cylinder and cone (default 6)
- nocap** do not put caps on long cylinders and cones
- linesmooth** smooth lines and text
- specpz** specular highlights take the view direction parallel to the z-axis
- fastselect** object interaction with the mouse does not use occlusion
- noisover** do not check if the cursor is over a clickable object
- maxtexture** maximum texture size (default 2048)
- samplerate** audio sample rate (default 22050)

Options that only influence the performance:

- amax** maximum size of GL arrays (default 10000)
- cull** minimum number of optimized vertices in objects for culling (default 250)

Options for debugging:

- dump** dump the scene
- dumpall** dump the scene including coordinates
- dumptree** dump the scene tree
- notrans** do not pre-transform the scene
- nolist** do not use GL display lists
- nozsort** do not z-sort polygons in transparent objects

- notexture** do not use textures
- single** do not use double buffering
- transparent** set Material transparency smaller than 0.5 to 0.5, makes the whole scene transparent

Additionally the GLUT version has a -stereo option for stereo viewing and the MPK version has a -config option for specifying the MPK config file.

User interface

The left mouse button can be used to click on sensors in the scene, when there is no sensor under the cursor the scene can be rotated by holding the left mouse button down.

Keyboard:

'Esc'	quit
'v' and 'c'	switch to the next and the previous Viewpoint of the scene
'n'	switch to the next Navigation type (vrml default: WALK, EXAMINE, FLY, NONE)
'w' and 's'	WALK, FLY: accelerate forward and backward EXAMINE: move the Viewpoint closer and further away
'a' and 'd'	WALK, FLY: accelerate to the left and the right EXAMINE: rotate the scene left and right
keypad '+' and '-'	increase and decrease the speed of the vrml time
keypad '*'	reset the speed of the vrml time to real time
'Backspace'	reset vrml time to 0, might not reset the scene completely to its original state
'Enter'	start the vrml time, only used when vrmlview has been started with -stop

The CAVELib version only supports the 'Esc', 'Backspace' and 'Enter' keys.

CAVE wand controls:

joystick forward and backward	move in the direction in which the wand is pointing, if the wand beam turns green it intersects a sensor
joystick left and right	rotate around the vertical axis
left button	activates sensors in the scene
middle button	see 'v' above
right button	see 'n' above

Node support

Currently 51 of the 55 VRML97 node types (www.web3d.org/technicalinfo/specifications/vrml97/part1/nodesRef.html) are supported.

Not supported are: Anchor, Collision, Extrusion, MovieTexture.

Additionally the new ReflectionMap node is supported, which can be used instead of an ImageTexture.

Most, but not all, routing to exposedField's (*set_...*) is implemented. Routing from exposedField's (*...changed*) is only implemented for *rotation* and *translation* in Transform.

The following nodes are not fully supported:

BackGround	skyAngle is not supported
FontStyle	not all alignment options are supported
Script	does not support SFNode and MFNode as field or event; Browser object functions and object functions are not implemented

Files with unsupported nodes are read correctly, but some functionality might be missing.

hpv.service.rug.nl

The binaries are:

- GLUT version: /usr/people/hess/vrmlview/glut/vrmlview
- MPK version: /usr/people/hess/vrmlview/mpk/vrmlview
- CAVELib version /usr/people/hess/vrmlview/cavelib/vrmlview

For a single screen the GLUT version is optimal. Use the CAVELib version for the Reality Cube and use the MPK version for the Reality Theatre. MPK configuration files for the Reality Theatre are:

- /usr/people/hess/vrmlview/mpk/mpk_th.mono
- /usr/people/hess/vrmlview/mpk/mpk_th.stereo

The manual for the VRML viewer

Bibliography

- [1] A-S. Axelsson, A. Abelin, I. Heldal, R. Schroeder, and J. Wideström. Cubes in the Cube: a comparison of a puzzle-solving task in a virtual and a real environment. *Cyberpsychology & Behavior*, 4(2), 2001.
- [2] J. Bohannon. BIOINFORMATICS: The Human Genome in 3D, at Your Fingertips. *Science*, 298(5594):737–, 2002.
- [3] V. Coors and V. Jung. Using VRML as an interface to the 3D data warehouse. In *Proceedings of the third symposium on Virtual Reality Modeling Language*, Monterey, California, United States, 1998. ACM Press, New York, New York, United States.
- [4] A. van Dam, A.S. Forsberg, D.H. Laidlaw, J.J. LaViola Jr., and R.M. Simpson. Immersive VR for scientific visualization: a progress report. *IEEE Computer Graphics and Applications*, (6):26–52, November/December 2000.
- [5] G. Domik, C.J.C. Schauble, L.D. Fosdick, and E.R. Jessup. Tutorial: Color in scientific visualization. University of Colorado, High Performance Scientific Computing Group, 1999.
- [6] M. Dröge. Numerical simulation of cold store air screens. Master’s thesis, 2000. University of Groningen.
- [7] G. Fekken, A.E.P. Veldman, and B. Buchner. Simulation of green-water loading using the Navier-Stokes equations. In *Proceedings 7th International Conference on Numerical Ship Hydrodynamics*, pages 6.3–1–12, Nantes, July 19–22, 1999.
- [8] L. Gamberini. Virtual Reality as a new research tool for the study of human memory. *Cyberpsychology & Behavior*, 3(3), 2000.
- [9] D. Isham. Developing a computerized interactive visualization assessment. *The Journal of Computer-Aided Environmental Design and Education (currently named Journal of Design Communication)*, 3(1), 1997. <http://scholar.lib.vt.edu/ejournals/JCAEDE/v3n1/>, South Dakota State University Apparel Merchandising and Interior Design, Brookings, SD 57007, isham@brookings.net.
- [10] E. Loots. *Fluid-Structure Interaction in Hemodynamics*. PhD thesis, University of Groningen, 2003.
- [11] A. Modi. *Real-time visualization of aerospace simulations using computational steering and beowulf clusters*. PhD thesis, Pennsylvania State University, 2002. www.personal.psu.edu/faculty/l/n/lnl/theses/Modithesis.pdf.

- [12] F.A. Nielsen and L.K. Hansen. Experiences with Matlab and VRML in functional neuroimaging visualizations. Princeton, New Jersey, April 2000. Visualization Development Environments 2000 Proceedings.
- [13] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall PTR, Upper Saddle River, New Jersey, 1996. www.vtk.org.
- [14] P.M. Trivailo. Matlab in building virtual reality worlds: animated dynamics of elastic space structure and advanced robotic systems. <http://www.ceanet.com.au/mluserconf/papers/Trivailo.pdf>, November 2000. Department of Aerospace Engineering, RMIT University, Melbourne, Australia.
- [15] B.K. Wiederhold. Virtual Reality in the 1990s: what did we learn? *Cyberpsychology & Behavior*, 3(3), 2000.