TEKNILLINEN KORKEAKOULU
Sähkö - ja tietoliikennetekniikan osasto

Markku Mantere

# Visualization of Flow Data in Photo-realistic Virtual Environment

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi
diplomi-insinöörin tutkintoa varten Espoossa 4.6.2001

Työn valvoja Professori Tapio Takala
Työn ohjaaja Matti Gröhn

**Author:** Markku Mantere
**Name of the Thesis:** Visualization of Flow Data in Photo-realistic Virtual Environment

**Date:** June 4th 2001 **Number of pages:** 91

**Department:** Department of Electrical and Communications Engineering
**Professorship:** Tik-111 Interactive Digital Media

**Supervisor:** Professor Tapio Takala
**Instructor:** MSc Matti Gröhn

Virtual reality technology has been adopted in many different fields and new application areas are searched continuously. At the moment virtual reality has been applied separately for instance to scientific visualization and illustration of architectural spaces. In this work, a photo-realistic room model and a visualization of an air flow inside the room has been combined. The integrated illustrative three-dimensional model is presented within an immersive virtual environment.

The first part of the work covers scientific visualization and virtual reality implementation techniques. The visualization review begins with a discussion about human percepion of visual information and proceeds with an introduction to three-dimensional visualization. The focus is on illustration of a flow data produced as a result of a computational simulation. The flow visualization techniques utilizing all three dimensions are discussed and many examples of different graphical elements are presented.

Virtual reality is examined from technical solutions point of view. The features having effect on the quality of a virtual experience are discussed and three different commonly used display techniques are introduced. The hardware of Experimental Virtual Environment -facility at Helsinki University of Technology is given as a detailed example.

The implementation of a visualization software is described in the applied part of this thesis. Discussion covers the evaluation of different software tools, the tool selection process, and a detailed description of the design principles and implementation of the software. The different visualization solutions are also justified in this part. In the implementation, the real-time system requirements and utilization of all three dimensions have been taken into account.

Finally, the results and their meaning are discussed and the performance of the implementation is evaluated. The applied part successfully integrated the room model and the flow visualization in an interactive virtual environment.

**Keywords:**
virtual environments, virtual reality, flow visualization, CFD, 3D, computer graphics

**Tekijä:** Markku Mantere
**Työn nimi:** Visualization of Flow Data in Photo-realistic Virtual Environment

**Päivämäärä:** 4. 6. 2001 **Sivumäärä:** 91

**Osasto:** Sähkö- ja tietoliikennetekniikan osasto
**Professuuri:** Tik-111 Vuorovaikutteinen digitaalinen media

**Valvoja:** Professori Tapio Takala
**Ohjaaja:** FK Matti Gröhn

Keinotodellisuustekniikkaa on viime vuosina otettu käyttöön monilla eri aloilla ja sille etsitään edelleen jatkuvasti uusia sovellusalueita. Toistaiseksi keinotodellisuutta on sovellettu erikseen muun muassa tieteelliseen visualisointiin ja arkkitehtonisten tilojen esittämiseen. Tässä työssä on yhdistetty todenmukaiselta näyttävä huonemalli sekä huoneen sisällä tapahtuvan ilmavirtauksen visualisointi. Työssä yhdistetty havainnollinen kolmiulotteinen malli esitetään sisäänsä sulkevan ja kuvan syvyysvaikutelman toteuttavan EVE-keinotodellisuuslaitteiston avulla.

Työn ensimmäisessä osassa tarkastellaan tieteellistä visualisointia ja keinotodellisuuden toteuttamistekniikoita. Visualisointikatsaus alkaa kuvallisen informaation havainnoimisen perusteista ja esittelee kolmiulotteisen visualisoinnin periaatteita. Lähemmin tutustutaan laskennallisin menetelmin tuotetun virtausdatan esittämiseen kolmiulotteisuutta hyödyntäen sekä esitellään useita virtausdatan visualisointiin soveltuvia graafisia elementtejä.

Keinotodellisuutta yleisesti käsittelevässä osassa paneudutaan virtuaalisen kokemuksen laatuun vaikuttaviin teknisiin ratkaisuihin sekä esitellään kolme yleisintä keinotodellisuuslaitteistoissa käytettävää näyttötekniikkaa. Tarkempana esimerkkitoteutuksena käydään läpi TKK:n EVE-laitteisto.

Työn soveltavassa osassa kuvataan visualisointiohjelmiston toteutus. Kuvaus kattaa eri ohjelmistotyökalujen vertailun, valintaperusteet sekä tarkan selvityksen ohjelmiston suunnitteluperusteista ja rakenteesta. Myös toteutuksen yhteydessä tehdyt visualisointiratkaisut perustellaan. Toteutuksessa on kiinnitetty erityistä huomiota reaaliaikajärjestelmille asetettaviin vaatimuksiin ja kolmiulotteisuuden hyödyntämiseen.

Kirjallisen osan lopussa pohditaan työn tuloksia ja merkitystä sekä arvioidaan toteutetun järjestelmän suorituskykyä. Työn soveltavassa osassa on tavoitteiden mukaisesti yhdistetty onnistuneesti huonemalli ja siihen liittyvä virtausvisualisointi.

**Avainsanat:**
keinoympäristöt, keinotodellisuus, virtausten visualisointi, CFD, 3D, tietokonegrafiikka

# Acknowledgments

After many years of active and semiactive studies, my master's thesis is ready.

Many people from different organizations have had influence to my work but I would like to express my gratitude especially to following contributors:

- My supervisor, Professor Tapio Takala, for making the VR research in TML possible.
- My instructor, Matti Gröhn, for valuable guidance and for pushing the project ahead.
- Mikko Laakso, my colleague on the BS-CAVE project for sharing with me the pleasant agony of working on master's thesis.
- Seppo Äyräväinen, my another co-worker for providing a lot of comments and company.

I would also like to thank my parents, Marjo and Raimo Mantere, for their support on every phase of my life.

And finally, thanks to Marika, the challenging love of my life.

Otaniemi June 4th 2001

Markku Mantere

# Contents

# List of Abbreviations

| | |
|---|---|
| 2D | Two-dimensional |
| 3D | Three-dimensional |
| 3DSMax | 3D Studio Max - 3D animation and modeling software |
| AI | Artificial Intelligence |
| ALSA | Advanced Linux Sound Architecture |
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| ASCII | American Standard Code for Information Interchange |
| AVS | Advanced Visual Systems - visualization software |
| BMP | Bitmap file format |
| BS | Building Services |
| CAD | Computer Aided Design |
| CAVE | Cave Automated Virtual Environment, registered trademark of University of Illinois Board of Trustees |
| CFD | Computational Fluid Dynamics |
| CRT | Cathode Ray Tube |
| DOF | Degrees Of Freedom |
| EVE | Experimental Virtual Environment |
| FEM | Finite Element Method |
| FOV | Field of View |
| GUI | Graphical User Interface |
| HMD | Head Mounted Display |
| HUT | Helsinki University of Technology |
| IEEE | Institute of Electrical and Electronics Engineers |
| IRIX | Operating system used in SGI's computers |
| IV | Open Inventor file format |
| JVP | VR Juggler Virtual Platform |

| | |
|---|---|
| LAN | Local Area Network |
| LCD | Liquid Crystal Display |
| MS | Microsoft Corporation |
| NASA | National Aeronautics and Space Administration |
| PC | Personal Computer |
| PFB | IRIS Performer BINARY database format |
| RAM | Random Access Memory |
| SGI | Silicon Graphics Incorporated |
| SID | Spatially Immersive Display |
| TCL | Tool Command Language |
| VMD | Virtual Model Display |
| VR | Virtual Reality |
| VRML | Virtual Reality Modeling Language |
| VTK | Visualization ToolKit |

# List of Figures

# Chapter 1

# Introduction

Human is a visual animal. We get a major part of the information from the surrounding world by visually observing it. It is also said that a picture is worth a thousand words. True or not, it is usually much more efficient to explain some matter to another person with a picture than using just words and numerical expressions. The power of visualization is based on this fact about human learning and insight mechanism. Human is also an economical animal, who wants to minimize the consumption of time and effort when aiming at any goal. If learning or comprehension is that goal, it is obvious that the material to be processed is illustrated as pictures, whenever possible. No wonder that the evolution of writing began anciently from descriptive symbols like Egyptian and Mayan hieroglyphics.

Since those ancient times, sciences have developed tremendously and we are able to product huge amounts of different kinds of information. Fortunately, techniques for processing different abstract information into more illustrative representations have also developed as the amount of data has increased. As a result of computerized numeric methods, more and more different problems can be solved by using simulations. Often the interpretation of the results must be done by examining vast numeric data masses. Visualization of the data is required to help, or even to make possible the actual understanding of some phenomenon. The visualizations have been constrained to paper and computer screen for a long time, providing only two-dimensions even for three-dimensional visualizations. But improvement is on the way.

Along with other technologies, the display techniques for computer generated images have also advanced to a level where it is possible to create very realistic three-dimensional experiences. If three-dimensional imaging is implemented to be really immersive, and is enhanced with sophisticated interaction techniques, the concept can be referred as virtual reality. By using these techniques, the reflection of real world, presented as an ordinary two-dimensional picture, can be extended back to the original three dimensions.

A natural development trend is to integrate visualization of interesting information and vir-

tual reality techniques to utilize all three dimensions. This way it is possible to provide the user with the most natural environment to observe characteristically three-dimensional information, such as architectural designs or simulations of many natural phenomena.

The Experimental Virtual Environment[1] (EVE) is a high quality virtual reality facility at Telecommunication Software and Multimedia Laboratory[2] (TML) at Helsinki University of Technology[3] (HUT). It is dedicated for the research of basic virtual reality technology and advanced visual and aural applications. Recently in Finland especially the building industry has indicated its interest to take part in developing new ways to process and illustrate building services (BS) information.

From this basis, the laboratory and a few representatives from the industry started a research project, which aimed to develop virtual reality techniques and applications for the building industry. The project was supported by the National Technology Agency of Finland. This project, called BS-CAVE, is introduced more closely in next section. This thesis concentrates on one partial objective of the whole project, which is to integrate a photo-realistic room model with a three-dimensional visualization of normally invisible air flow, and to present them together in the EVE.

Architectural models suit well and intuitively to virtual environments and they are a well understood application area. Intuitiveness refers to a the fact that people are used to see real buildings. They have a preconception that the floor is below, the ceiling is above, and how wide a door should be, etc. They also have an understanding of the relations for example between windows and doors. Besides the architecture of the space itself, it is often valuable to combine other information types in the same model. A scientific visualization is more understandable if it is presented in a familiar context. In this project, the flow visualizations will be presented in a room model.

## 1.1 BS-CAVE project introduction

This work is done as a part of a project called '3D Visualization of Building Services in Virtual Environment', BS-CAVE in short. The project was established to develop methods and techniques for building services visualization, especially in a virtual reality environment. The project concentrates partly on the integration of indoor climate and ventilation conditions visualization and photo-realistic room models, and partly on sophisticated three-dimensional (3D) navigation methods in a virtual environment.

The project is carried out mainly in the Telecommunication Software and Multimedia Labor-

[1] <URL:http://eve.hut.fi/> 4.6.2001
[2] <URL:http://www.tml.hut.fi/> 4.6.2001
[3] <URL:http://www.hut.fi/> 4.6.2001

atory at Helsinki University of Technology. Besides the laboratory, the other participants are Oy Halton Group Ltd[4], engineering office Olof Granlund Oy[5], and VIVA-group. The latter is a group of luminaire manufacturers. A major part of the funding comes from Tekes - The National Technology Agency[6].

BS-CAVE is one of the largest application development projects in the EVE so far. The project has three objectives. In the project application they were defined as follows:

- Determination of a reliable and (semi)automated information transmission path from the companies' model databases to the virtual room at HUT.
- Transmission of flow data to the virtual room and developing new ways to visualize it.
- Development of new interaction techniques appropriate for visualization of building services in a virtual environment.

The focus of this thesis is on the issues related to the second objective, the visualization of flow data. From the practical point of view, the formulation referring to 'new ways to visualize' should be understood more like 'suitable ways to visualize'. The resources in this project are limited, and the research and development of totally new visualization algorithms and methods are out of the scope of this thesis. In a nutshell, the work consists of defining practical methods to transfer flow data from a computational fluid dynamics (CFD) solver software to a visual form in a realistic virtual environment to the EVE. To achieve the objective, the required software is implemented in the laboratory. Olof Granlund Oy provides the flow data and the photorealistic room model. The company works in a Windows personal computer (PC) environment while the system in the EVE is built on UNIX operating system.

## 1.2 Visualization problem statement

The definition and delimitation of the problem discussed in this thesis is straightforward because the work is done according to the initial project objectives. The overall problem is thus the transmission of flow data to the virtual room, and developing new ways to visualize it. The objective can be broken down to three tasks.

The first task is to transfer the flow data from a CFD solver software to visual form in the virtual environment at HUT. To accomplish this task, the data must first be converted to a suitable format and then processed properly to generate the actual geometric visualization elements. The solution of the visualization problem includes also decisions on how to

[4]<URL:http://www.halton.fi/> 25.5.2001
[5]<URL:http://www.granlund.fi/> 25.5.2001
[6]<URL:http://www.tekes.fi/> 4.6.2001

visualize something invisible by its nature, such as air flows in a room space. In the implementation of the visualization software, a few existing visualization techniques are applied. The challenge of developing totally new visualization elements are, however, left out of the current scope. The visualization problem in this project is not only technical but also related to the true existence of all three dimensions. In the evaluation of different visualization element types, the focus will be on how they strengthen the experience of the space in a virtual environment.

The second task is to merge the photo-realistic model into the same system with the flow data visualization, and to combine them together. Problems related to different coordinate systems must be solved in this phase.

The third task is to represent the combined model in a high-end virtual environment providing the user with strong immersive experimence. The implementation should also utilize techniques providing the interaction with the environment. From the flow visualization point of view, the interaction here refers to the possibility to choose in a reasonable way, which visualization elements are seen in the scene at a time.

The focus in this work and thesis will be on the interactive flow visualization. The visualization will be adjusted particularly for 3D virtual environment. On a practical level the, objective is to design and implement a software matching the requirements of the project. The virtual environment at HUT and the development of the facility is also an important issue and closely related to this work. However, detailed discussion on this area is left out of the scope of this thesis.

## 1.3 Criteria

As in every project with a predefined time frame, criteria must be defined to be able to judge if the current state of the project corresponds to the initial objectives. In this project, most of the criteria are not numerically measurable. Because only very few accurate numerical criteria can be used, the evaluation must be done mainly based on the users' qualitative comments. The only numerical key figure, which will be measured in this project, is the frame rate of the system. The following essential requirements help to outline the overall objective.

#### Visualization

The goal is to visualize flow data. The visualization will be presented in a 3D virtual environment and thus there are special requirements for the visualization elements. The essence of the elements should utilize the spatial features of the display technique. The visualization

elements placed in the scene are not just for entertainment, but to give the user a visual tool for better insight. At the end of the project it is important to evaluate if the visualization actually fulfills its function.

Because the flow visualization is presented in a real 3D environment, the essence of the visualization elements used, should strongly support the user with a sense of space. The requirements for the use of different geometric elements differs from conventional desktop applications. An optimal visualization element is truly three-dimensional, so that its spatial behavior will help the user to localize it in the space.

### Usability

From the user's point of view, the different phases where the information is transferred to the system and processed further, are evaluated by the convenience of the task. The goal is to minimize the amount of manual work. The information refers to the photo-realistic models, and the results from the CFD solver software – the flow data. The user should not consider the task too complicated, and there should not be a need to edit any data manually. In every phase of the visualization process, the usability should be kept in a high level. The required visualization result must be achieved without unnecessary effort. In other words, using software tools. The tools may be implemented within the project or adopted from other sources.

Because it is obvious that different flow phenomena should be visualized with different geometric elements, it would be convenient for the user to be able to change the visualization smoothly and interactively while the system is running. Being able to vary the visualization in the scene allows the user to observe the behavior of the flow from different aspects, and thus to better associate different pieces of information.

Another criterion for the system implementation is to provide means to transfer the resulting visualization to a normal PC environment. This way the visualization can also be used outside a high-end virtual reality environment.

### Immersion and the virtual experience

The resulting scene in a virtual environment should be understandable. The chosen visualization elements should fit together with the photo-realistic model, and the geometry presenting the space itself will remain recognizable after the addition of the flow visualization. The visualization elements used should also themselves have a clear meaning to the user. The user will be better aware of his surroundings if the scene is kept relatively clear and the flow visualization can be done using only few meaningful objects. If the awareness of the environment is continuous, the immersion will be stronger.

The immersion and the sense of space also depends on the amount of depth cues the visualization elements are able to provide. Hence the visualization should support both the flow information and the three-dimensional presentation.

## 1.4 Thesis Organization

This thesis is organized in six chapters discussing various topics related to the BS-CAVE project. The main topics are flow visualization, virtual reality, and the actual software implementation. This first chapter introduced the BS-CAVE project, its overall objectives and the criteria, which will be used in the evaluation of the results.

Chapter 2 describes the principles related to scientific visualization in general and takes a closer look at the visualization of 3D flows. Chapter 2 also discusses the numerical simulation methods, which are used to produce computational flow data.

Chapter 3 shortly discusses the techniques used to create immersive virtual reality. The hardware setup used at Helsinki University of Technology is described in detail.

Chapter 4 describes the implementation phase from component selection to internal functionality of the implemented software. The software component selection is reviewed in detail to give the reader an impression of how immature the available virtual reality (VR) software systems still are. The detailed requirements for a virtual reality visualization system are discussed before explaining the implementation. The chosen visualization elements and techniques are also described.

Chapter 5 presents the results and evaluates how well they correspond to the requirements and expectations. The performance capability of the system is discussed based on the measured frame rates. The problems raised during the project are also presented in Chapter 5.

Finally, Chapter 6 draws conclusions from this thesis and the current status of the implementation. The implementation is also reviewed from a future research point of view. A possible follow-up project is sketched.

# Chapter 2

# Scientific visualization

## 2.1 Visualization in general

### 2.1.1 Purpose of visualization

In general, information visualization aims at making things visible. As mentioned in previous chapter, human learning capabilities, pattern recognition and piecing things together, can easily be amplified by using pictures and other visual material. Therefore visualization of different information makes things easier for us and, in many cases, helps us to see things in new ways. Mathematics and geodesy, for example, have developed numerous visual techniques to make tasks more efficient. Nomographs speed up calculations and navigation charts actually enable seafaring. Even the simplest visual methods extend the working memory and help to manage the information needed. Other senses like hearing and sense of touch, can also be used to present information. These techniques are correspondingly called sonification [Kramer et al. 1999] and tactilization [Card et al. 1999], but they are not discussed in this thesis.

The history of actual data visualization derives from the 18th century [Card et al. 1999]. Before the computerized era different graphs and drawings have been the most usual method of visualization. An example of other kinds of visualizations is the use of wind tunnels. In these facilities, aerodynamical properties of objects are studied in an air flow by using smoke or threads to indicate the behavior of the flow around the object. At the sixties when the computers were matured enough to produce graphical output, visualization began to develop to the direction where it is today. In twenty to thirty years the field of visualization matured so that Institute of Electrical and Electronics Engineers (IEEE) established its own annual visualization conference in 1990. Today the applications cover many areas from astronomy to medical science and from engineering to stock market analyses.

Modern visualization is defined in [Card et al. 1999] as follows: ”The use of computer-supported, interactive, visual representations of data to amplify cognition.” The purpose is not to produce a nice picture, but to bring out the essential of a data set. As it is said about computers that “the purpose of computation is insight, not numbers”, about visualization it could be said that “the purpose of visualization is insight, not pictures”. Further, the ultimate task where the information visualization is just a subtask, the exploration of phenomena and information sharing between professionals of different areas is, in a word, insight.

### 2.1.2 Amplifying cognition

It has been found that vision is related to several different cognition mechanisms. The mechanisms can also be applied to the scientific visualization. Numerous authors have recently proposed these mechanisms related to information visualization. In [Card et al. 1999], the mechanisms have been gathered and classified under six topics.

#### Increased resources

Human visual and perceptual systems are able to combine spatial understanding with wide dynamics in sensing the environment. Sense of sight can adapt from almost pitch-black to the brightest sunlight. Vision can be considered as a high-bandwidth spatially hierarchical communication channel. Low level visual tasks can also be processed in parallel, utilizing just simple perceptual operations. As mentioned earlier, visualization has expansive effects by extending the working memory by storing a lot of information in a quickly accessible form.

#### Reduced search

The time spent on searching can be reduced by grouping the information correctly. The ordering of the information in compact forms, also has the same effect. Labeling or other additional metadata relating to data localization and grouping may be avoided by using the spatial feature of visualization.

#### Enhanced recognition of patterns

When the information is illustrated in images, the user may do searches primarily by recognizing familiar patterns, instead of recalling the whole information mass. In a visualization process, the information may be organized and parts of it may be emphasized so that the essential content will be recognized by humans natural pattern recognition characteristics.

The behavior of data has three levels: value, relationship, and trend. Any of them can be used as a basis of the data organization.

**Perceptual inference**

Sometimes when a problem is illustrated as an image, the nature of the problem or even the solution seems immediately obvious. This is called perceptual inference.

**Perceptual monitoring**

By using vision, it is possible to observe a large number of different elements or potential events, and still notice anomalies very fast. As an example, a pilot is trained to monitor many gauges simultaneously and to react when one of the values deviates from the default.

**Manipulable medium**

By definition, computerized visualization provides a possibility for interaction. In other words, the user may change the parameters and have effect on the visualization. By these means, it is possible to find the most valuable way to visualize the the set.

### 2.1.3 Reference model for data visualization

In [Card et al. 1999] a simple reference model is proposed for data visualization. A formal model will clarify the discussion and the definitions of the relatively young field of visualization. It also helps in comparison of different systems.

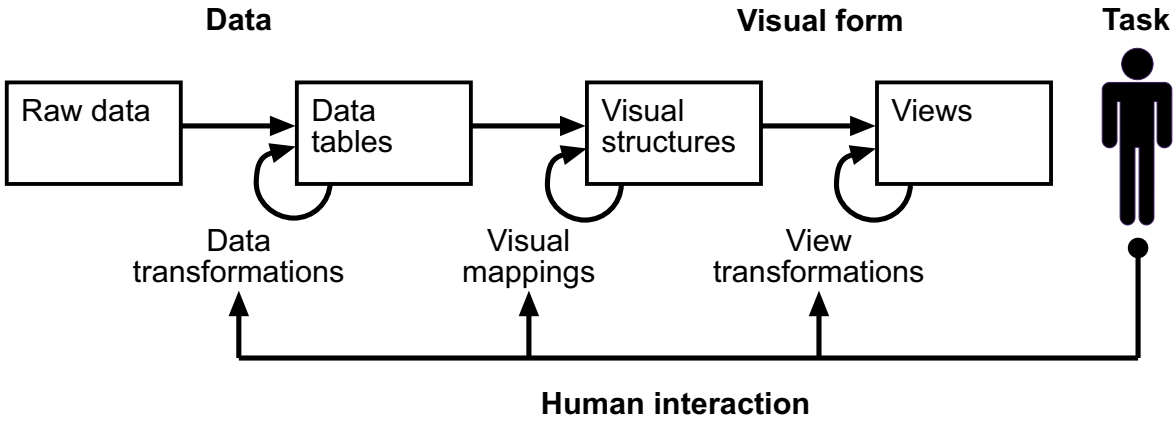



Figure 2.1: Reference model for visualization [Card et al. 1999].

Figure 2.1 illustrates the phases of a visualization process from raw data to the views shown to the user. The basic idea is that the original data is transformed or filtered in different ways until it is mapped as a graphical representation. The raw data is first organized in data tables, where the required relations between data units are defined and metadata is added. This is especially important for abstract data. Data from natural sources often has explicit spatial relations. In the next phase the data tables are used as a basis of the generation of visual elements. This is the most essential part of the visualization process. Here the data is transformed from numerical or textual representation to graphical objects and it is vitally important to choose such elements that can be processed effectively by human vision. The last phase, view transformations, adjusts the geometric representation and sets the parameters defining position, scaling, and clipping before the actual image rendering. Small circular arrows are used to emphasize that each phase can consists of actually multiple chained operations fo the same category.

The arrows back to the process from the user having a task to fulfill, represent the interactive characteristics of visualization. The process can be controlled in every phase to achieve an appropriate result.

### 2.1.4 Computerized visualization

It is not a trivial task to produce an effective visualization. Traditionally visualization systems have been very complicated and have required special expertise. The researchers have had to be specialists not only on their own area, but also on programming and display techniques. This has meant that visualization has required co-operation between researchers and computer engineers, or that the researcher of a particular field must have learned these skills himself. While the visualization has matured, people on the field have began to demand easier tools to create sophisticated visualizations, without having to be trained computer programmers. The ultimate goal should be that a scientist could say to the system: ”Give me a visualization of this data to understand phenomenon X.” Today, the challenge is still on a level of choosing a right size, placement, color and labeling for the graphical elements when creating a proper visualization. Still, there is often need for co-operation of experts with different backgrounds. The psychological and cognitive sides of visualization and the importance of correct emphasizing is discussed widely in [Keller and Keller 1993] with many examples.

Automated visualization has been studied and a system called AutoVisual[1] has been introduced. The idea is to utilize artificial intelligent (AI) as part of the visualization design process to avoid unnecessary repetition of time-consuming routines [Beshers and Feiner 1994].

[1] <URL:http://www.cs.columbia.edu/graphics/projects/AutoVisual/AutoVisual.html> 25.5.2001

Other approaches to creation of new data visualization tools and development of their usability, is discussed in [Foley and Ribarsky 1994].

Advanced Visual Systems (AVS)[2], Iris Explorer[3] and OpenDX[4] represent modern trend in visualization software systems. They all implement the idea of a modular visualization process, where data flows through different phases according to the module composition. User builds visualization pipelines ou of many different modules by connecting them together. Typically there are modules for data reading, processing, combining, and rendering. Every module has inputs and outputs for connections to other modules. Modules may also have parameterization features for correct adjustment. Figure 2.2 presents OpenDX software in use. The pipeline is shown visually on the left. The actual Earth-related visualization is on the right, and the smaller windows on the bottom are used for module control.

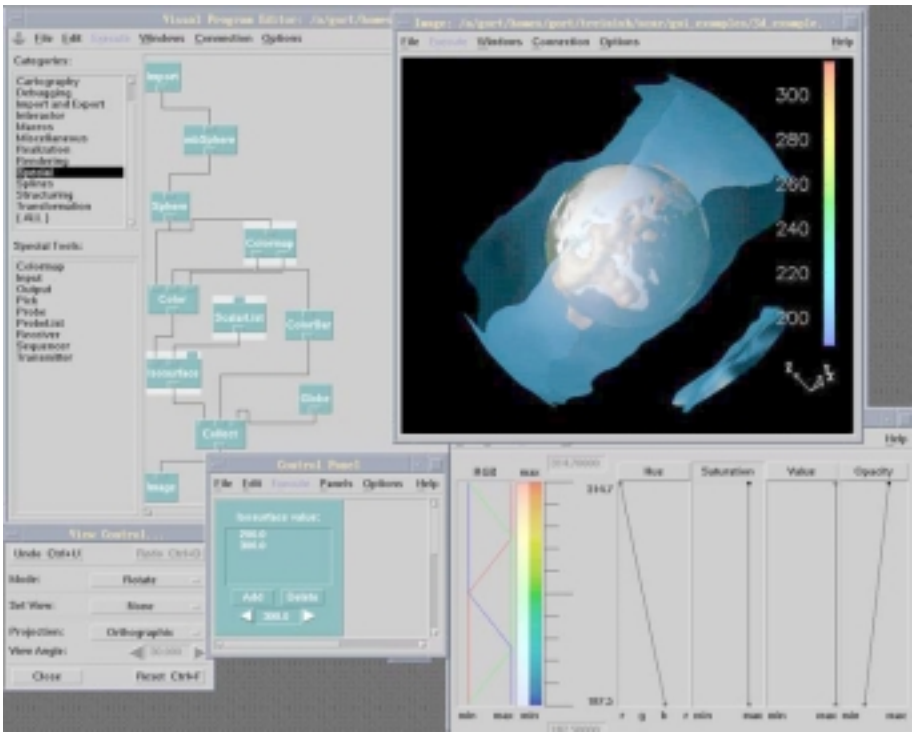



Figure 2.2: OpenDX in use. Raw data is produced by National Aeronautics and Space Administration (NASA).

## 2.2 3-D visualization

The power of 3D visualization is in its spatial characteristics. It enables easy mapping of data sets gathered from many natural phenomena. If the data is originally three-dimensional

[2]<URL:http://www.avs.com/> 25.5.2001
[3]<URL:http://www.nag.co.uk/Welcome_IEC.html> 25.5.2001
[4]<URL:http://www.opendx.org/> 25.5.2001

such as measurement values from the atmosphere or simulated flows in a container, the most obvious visualization also uses three dimensions. Various abstract data may also be mapped in three dimensions by defining certain relations to the different quantities of the data set. It is even possible to present n-dimensional data sets by using a technique called 'Worlds within worlds', which is based on nested heterogeneous coordinate systems [Feiner and Beshers 1990]. In brief, 3D visualization suits for measured real data, data generated by simulations, and even multidimensional abstract data.

Another issue is the technical approach: How will the visualization be implemented? The rendering may be based on surface graphics or volume graphics. The two techniques can be understood as extended versions of vector graphics and raster graphics. The former defines points, lines, and surfaces as geometric elements in an empty space. The latter divides the space in small cubical volume units, voxels, and defines one or more values for every unit [Kaufman 1994]. Although the data models differ, on both cases the images are finally rendered on raster displays. Both techniques have their pros and cons but the development trends have favored surface graphics. Volume graphics has so far succeeded best on medical visualization applications. The applied part of this thesis is implemented using surface graphics.

The view transformations (see Section 2.1.3) must be designed carefully, especially in the case of 3D visualization. The visualization objects in space may cover each other, so that essential information is hidden. Thus it is important to provide the user with a possibility to change the point of view, scaling, and clipping interactively. Specific coloring may be used to reveal additional information and relations in the data set. If the information density in the visualization is very high, an intentional distortion can be used to highlight a specific area in the data [Carpendale et al. 1997].

## 2.3 Visualization of flow data

### 2.3.1 Numerical methods and CFD

Mathematical models such as the difference method, control volume method, and finite element method, can be used to simulate different physical conditions concerning all kinds of materials. These numerical methods are based on the assumption that all matter builds up from an unbroken continuum [Koponen 2001]. For example finite element method (FEM) is a numerical computation method, which was originally developed for structural analysis of airplanes at the fifties. At the sixties the method was adopted in other application areas and was proven to be applicable also to computational fluid dynamics. Nowadays, the techniques can be used to verify many structural and functional aspects of different products,

without a need to build expensive mock-ups. Another benefit is that the numerical methods are flexible, allowing several alternative designs be tested in a relatively short time. Sometimes there may be a need for comparison of the results with real-world measurements, but on well known problems the methods are very reliable. In this thesis the visualized flow data is produced as a result of an office room air conditioning simulation utilizing numerical computation methods (see Section 4.5.2).

Although the actual computation phase between the methods varies, the overall process typically includes the same five steps. The first step is to define the basic geometry for the primary object and its environment. In the second step, the computation mesh will be generated by dividing the basic geometry in small units, cells. The density of the mesh is proportional to the computation accuracy and the quality of the mesh generation has a fundamental influence on the overall accuracy of the simulation process. The mesh or a 3D grid is built of triangles, cubes, tetrahedra or other geometric primitives. It is also typical that the density of the mesh varies along the object. Smaller details require a denser net and curved surfaces are often divided in smaller units than the flat parts of the object. The third step consists of parameterization of the model and the surrounding environment. Variables and initial conditions define for example materials, temperature, pressure and initial velocities and forces. These first three steps are usually called the pre-processing part of the CFD process.

The actual computation analysis and simulation is done on the fourth phase, also known as the solver part. If the initial parameters are not time-dependent and the computation converges, the process results in one steady stage. In case of a dynamic simulation, the results are produced for multiple time steps. The results are presented as data sets where each nodal point of the computation mesh contains a certain amount of vector[5] and scalar[6] values. Usually the problem is so complex that the solver software must utilize iterative methods. It is then possible that the computation will not converge and the initial conditions must be adjusted before re-running the simulation. The consumption of time taken by the computation is strongly related to the density of the computation mesh and the rapidity of the convergence. Also, additional time steps in time-dependent simulations increase the computation time and the amount of the resulting data.

After the simulation is done and the resulting data is stored in a computer file system, the fifth phase, the exploration of the results in a visual form begins. In the case of a CFD, the data consists of description of the object geometry and computation mesh, including values attached to the nodal points. These data sets are just masses of numbers, and they certainly need to be illustrated in a different form. This is where visualization comes forward. A proper visualization of large data sets is the only way for humans to understand the produced information. The CFD solver software may have integrated features for visualization. The

[5]Vector is here understood as a quantity, which presents both magnitude and direction.

[6]Scalar is here understood as a quantity presenting only magnitude.

fifth phase can also be done using a dedicated visualization software such as EnSight (see Sections 4.2.5 and 4.5.2). The visualization may be presented on a computer screen, printouts or even by means of virtual reality. The latter is the primary approach in this thesis.

### 2.3.2 The nature of flow data

Flow data is actually a collection of geometric definitions and different quantities. Flow data usually consists of a vector quantity for flow velocity and a few scalar quantities, for example temperature, pressure and density. The flow itself is an invisible phenomenon of substance transmission. In the nature it only becomes visible through moving matter, which is influenced by the energy of the flow. In computational flow data the properties of the flow are approximated by using a vector grid of a certain density. The behavior of an air flow can be illustrated in reality indirectly for example with wind vanes, dust and colored gases. In computerized visualization, animating particles is the most natural way of illustrating the phenomenon. However, the animation alone can not be used for accurate exploration of the whole data set. Therefore many artificial visualization techniques have been developed to illustrate flows [Post and van Wijk 1994]. The most commonly used techniques are introduced in the next section.

Every value in the data set has an explicitly defined position in the space. During the visualization, more values can be interpolated between the original nodal points, if required. In general, flows are 3D phenomena and they have a natural spatial mapping in a 3D visualization, but there are also special cases where the interest is in purely laminar, very small scale 2D surface flows.

One characteristic of flow data is that data sets can be huge. Largest data sets are reaching a magnitude of terabytes. It is obvious that large scale CFD simulations require a lot of computing power and time. At the moment it seems that CFD could spend all of the available resources in the near future, since one can always make the computation meshes denser or shorten the time steps, thereby producing an ever-increasing amount of data.

### 2.3.3 Techniques for visualizing 3D-flows

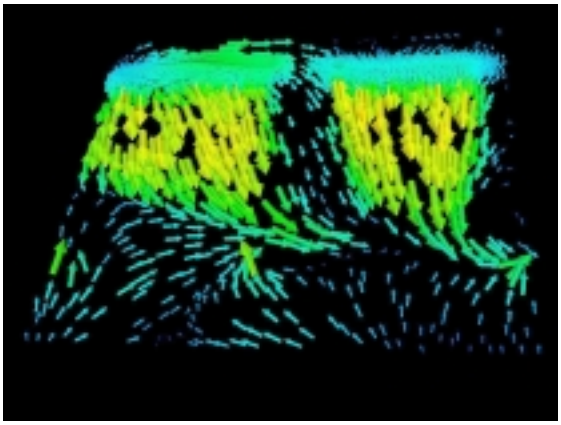
This section introduces shortly the most common visualization elements used to illustrate 3D flow data. The elements are presented in an order based loosely on the computational complexity of each technique. It should be noted that besides the geometry, the efficient 3D visualizations are based also on the right coloring, emphasizing and a proper view point. Thus choosing the right geometric elements for a visualization purpose does not cover the whole process, but is still an essential part of it.

Some of the elements introduced below were chosen to be implemented in the applied part of this thesis. The motivation of the selections and the chosen techniques (streamlines, stream tubes, animated particles, cross sections and isosurfaces) are discussed more at section 4.5.1, beginning from page 53.

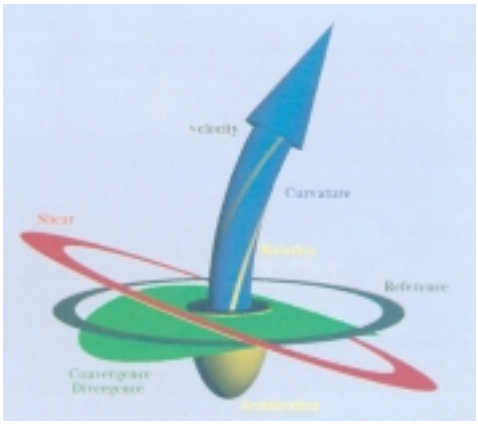
### Arrow plots

An arrow element is a traditional way to illustrate a vector quantity, both in 2D and 3D. Like a vector, an arrow has a position, a length (magnitude), and a direction. In flow data, each nodal point contains information in vector form about the velocity and the direction of the flow. The most obvious way to visualize it, is to draw an arrow on every nodal point [Post and van Wijk 1994]. However, if all the vectors of a data set are visualized together, the resulting image reminds more a hedgehog than a proper visualization. Because of the cluttering problems, arrow plots suit much better 2D than 3D. This is why in 3D visualizations, the arrows are usually drawn using a subset of all the vectors, e.g. the vectors on a cross section of the whole volume.

In Figure 2.3 (a) the arrows are drawn on a thin layer of a cubical flow volume. The length and the thickness of the arrows correspond to the magnitude of the flow velocity.



(a) Arrow plot.




(b) Probe for flow fields [de Leeuw and van Wijk 1993].

Figure 2.3: Vector field representation as arrows and a special glyph.

### Glyphs

Glyph is a general term for localized graphical objects, which represent the features of the data [Schroeder and Martin 99]. Glyphs may be placed among the other representations in

3D space to visualize the behavior of the data in that particular point, or near that point. Designing of glyphs is discussed throughly in [Foley and Ribarsky 1994]. Figure 2.3 (b) presents a probe-object proposed in [de Leeuw and van Wijk 1993] for local inspection of flow fields. With just one object, it is possible to represent six different properties of a flow: velocity, curvature, rotation, shear, convergence/divergence and acceleration. As it is seen, the interpretation of complex glyphs may require special expertise and knowledge about the illustrated phenomenon.

Another application area for glyphs is annotation techniques [Loughlin and Hughes 1994]. While studying a flow visualization, the user usually wants to make notes of the observations. Different glyphs and interaction techniques can be used to make notes directly into the visualization. Using the notes, the user will be able to easily return to the earlier thoughts. Annotations utilize the concept of extended working memory mentioned in Section 2.1.2.

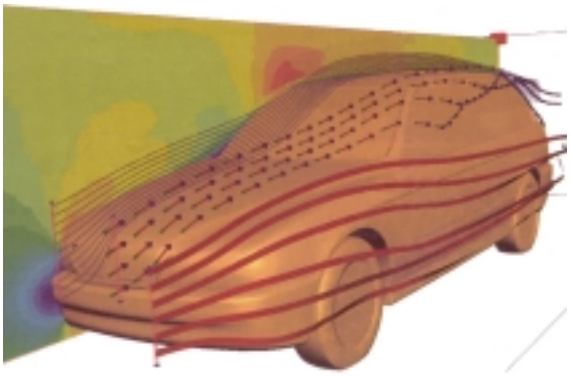
**Streamlines, stream ribbons and stream surfaces**

Visualization techniques based on particle traces in flow fields are very common. If a weightless particle is placed in a flow defined by a vector field, it is possible to integrate the path of the particle along the flow as a function of time. A basic algorithm for a simple case using discrete time steps is presented below in piece of pseudo code [Sadarjoen et al. 1998]:

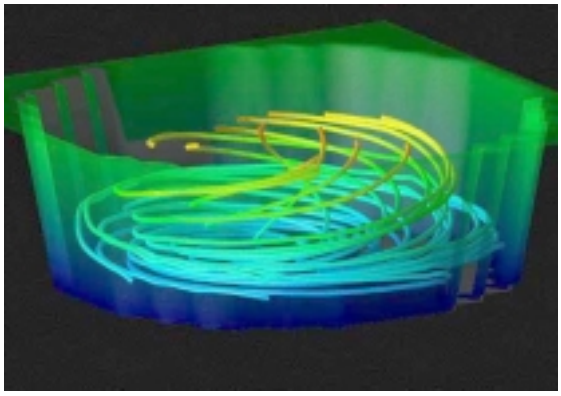
```
find cell containing initial position          (point location)
while particle in grid
      determine velocity at current position   (interpolation)
      calculate new position                   (integration)
      find cell containing new position        (point location)
endwhile
```

The resulting path may be visualized by using ordinary lines, ribbons or stream surfaces [Post and van Wijk 1994], [Schroeder and Martin 99]. Lines are thin objects without a diameter, illustrating only the path of the flow. Ribbons are like stretched lines having length and width but still lacking thickness. Besides the path, ribbons show also the vorticity of the flow. Stream surfaces are generated by cross sections of visualization elements moving along stream paths. For example, a circle produces a stream tube when it is moved along the particle trace. The two latter elements, ribbons and stream surfaces provide better depth cues to the user and thus they suit 3D visualizations better (see Section 4.5.1). The disadvantage with ribbons and stream surfaces is that they may overlap too much with the rest of the scene and prevent proper exploration of the visualization.

Figure 2.4 (a) illustrates streamlines and stream ribbons in a car aerodynamics application. Stream tubes in Figure 2.4 (b) are an example of stream surfaces.



(a) Streamlines and stream ribbons with arrow glyphs and cross section on background [Schulz et al. 1999].



(b) Stream tubes in a well from Australian Institute of Marine Science.

Figure 2.4: Visualization elements based on particle trace techniques.

The particle trace elements can be improved by dynamic coloring or additional texturing. The colors on the surface of the elements may illustrate the properties of other (scalar) quantity. Stream ribbons may be colored based on the temperature values on each nodal point along the path. Textures mapped on the objects strengthen the depth cues. Animated textures can be used to present the direction of the flow, for example in a case of stream tubes [Fuhrmann and Gröller 1998].

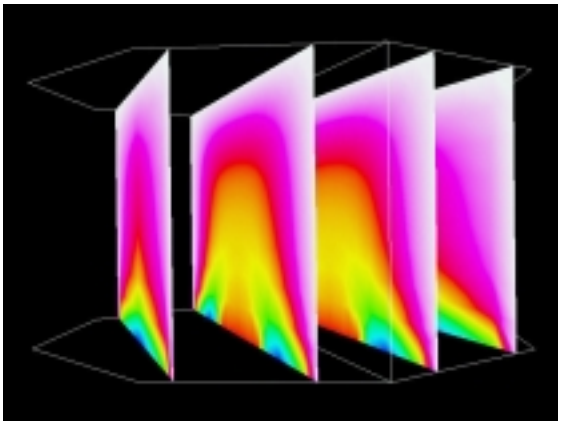
### Particle systems

The information about the paths of particles can also be used as the basis of an animation. It is effective to add the actual movement of the particles in the visualization (see Figure 4.14 on page 60). Animations are also the most realistic way to visualize fluid flows [Post and van Wijk 1994]. The animated particles may be rendered as simple geometric objects such as points or spheres, or even complex, dynamically changing glyphs.

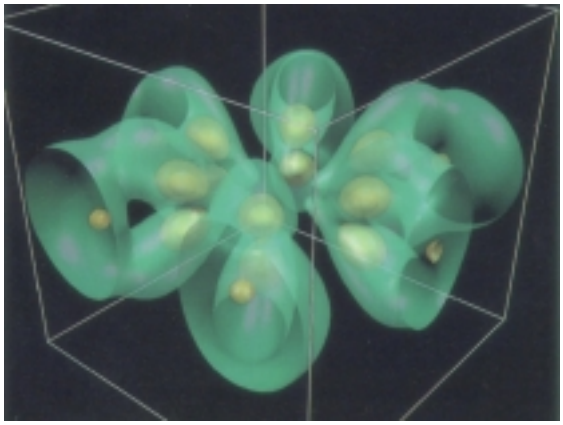
### Cross sections

Volumes containing information may be sliced to create cross sections [Schroeder and Martin 99]. The cutting may be done with any type of an implicit function, but the most used is an ordinary plane. It produces a flat sectional plane, which can be colored according to the information in the data set. In Figure 2.5 (a) there are four cross sections colored according

to the temperature in the volume. In Figure 2.4 (a) the cross section on the background of the car presents the velocity of the passing air.



(a) Four cross sections.



(b) Isosurfaces presenting electric charge density of a benzene molecule [Ruokolainen and Gröhn 1996].

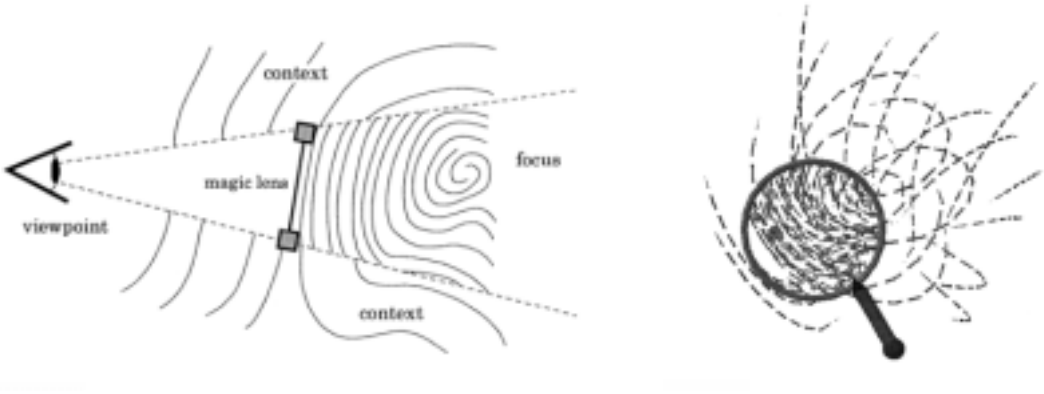
Figure 2.5: Plane and curved surface techniques in 3D visualization.

**Isosurfaces**

Isosurfaces are arbitrary shaped surfaces, where the magnitude of a chosen quantity is equal [Schroeder and Martin 99]. Important critical values of a quantity in a data set can be studied by using isosurfaces. One surface divides the volume in two regions. In the other side of the surface the value is bigger and in the opposite side smaller than the reference value on the surface. Figure 2.5 (b) presents a case of multiple isosurfaces on the same visualization. An isosurface does not necessarily have to be colored with only one color but can also be colored according to the magnitude of other quantities in the data set.

**Magic lenses and boxes**

The idea of magic lenses is the following. The user looks at the 3D scene through a special movable flat widget, which changes the view of the scene behind the lens [Napari 1999b], [Fuhrmann and Gröller 1998]. The changed view may contain more or less details, or present additional information that would not be visible without observing the scene through the lens. Multiple lenses can be used simultaneously on top of each other to create different feature combinations. Figures 2.6 (a) and (b) illustrates how magic lenses are used in 3D flow visualization.




(a) Volume defined by a magic lens.

(b) Focusing with a magic lens. Here the lens is used to add details in the flow visualization.

Figure 2.6: Magic lenses: Conceptual graph (a) and magic lens in use (b) [Fuhrmann and Gröller 1998].

The lenses can also change the view completely. One example would be an architectural application, where the user would be able to use magic lenses to see through walls to the next room, or to locate electric wiring inside the wall.

Magic box is a 3D variation of a magic lens. It provides a volume where the changes will take place. While the lenses are looked through and the changes take place everywhere *behind* the lens, the boxes are looked in and the changes happen only *inside* the box.

# Chapter 3

# Virtual reality

## 3.1 Techniques for creating immersion

Virtual reality is something unreal appearing real through the observer. In a nutshell, as it is understood in the context of this thesis, virtual reality is achieved basically by cheating the senses so that the observer interprets the various computer generated stimuli as a real phenomena. If the sensation appears true enough, the observer feels being a part of the artificial world. To be more precise, the user experiences that the artificial elements are part of his real world.

When the observed artificial stimuli appears to enclose the user inside the virtual world, the experience is said to be immersive. On its strongest form achieved today, virtual reality replaces the real world stimuli by visual, audible and tactile signals. Other senses, such as taste and smell, are nowadays usually ignored because of the lack of appropriate technology [Jalkanen 2000]. The immersion is not an on/off phenomena, but more like a gradually emergent and strengthening concept. The depth of the immersion is a sum of many features of the facility, which is used to create the virtual environment, and appealing properties of the particular application. The following paragraphs describe a few features that have significant importance on the quality of an immersive VR experience. This work concentrates on the most usual VR display types today, such as spatially immersive display (SID), virtual model display (VMD), and head mounted display (HMD). They are introduced in Section 3.1.1. Experimental and mostly conceptual techniques, like autostereoscopic displays, are left out of the scope of this thesis.

#### Performance

The overall performance of a VR system is perhaps the most important single factor in making the virtual environment immersive and credible [Bierbaum 1998]. The function of any

VR system is to provide the user an appropriate experience. If the system is not able to do this, it has failed in its main task. Interactivity and adequate frame rate on image generation, are core features of a virtual reality application. The user can not achieve an immersive experience, if the system performance is not high enough, and thus it does not answer the basic requirements. Besides that, in the worst case, the virtual environment will be unexploitable and inconvenient. In some conditions, low frame rate and delay in the responses may cause even motion sickness, which is also known as simulator sickness. The nausea phenomena related to delays in visual systems, are discussed in detail in [Allison et al. 2001].

**Stereo imaging**

An ordinary picture is flat. The real world is three-dimensional. Although flat pictures may have content that is interpreted to present something actually three-dimensional the picture itself is still just two-dimensional. In modern VR systems, the virtual world is presented using true 3D imaging. This is achieved by using a few key features of the human vision system and presenting two slightly different pictures, one for each eye. These features are referred as 'retinal disparity' and 'parallax' and are described in great detail with many other concepts related to stereo vision in [Lipton 1997]. Basically active stereo imaging is implemented either with time-shared multiplexed shutter glass technique, or by using a head mounted display (HMD) with small separate displays for each eye. The idea of shutter glasses is that while one eye is shut at a time, another eye sees a correctly projected image. In the next phase, the other eye sees slightly different picture and the other eye is shut. High-end VR utilizes active stereo techniques, but also passive techniques like polarization based image separation exist. However, they limit the user's movement and orientation because the separate images are seen either on a very narrow focus area, or only on a specific orientation.

Human vision system is very adaptive and it tries to form depth interpretation from almost any slightly different pictures shown to each eye. The drawback is that it is very easy to implement a system, which produces the intended stereographical images using an inappropriate geometric approach. Even though the resulting view may appear correct, and give the user a feeling of depth in the scene, incorrectly implemented stereo effect causes a bizarre feeling. In any case, the user's eyes are at least unnecessarily stressed while they try to adapt to an unnatural view, and even dysphoria have been observed. To avoid annoying experiences related to stereo vision, the images must be generated using two separate 'cameras' having parallel lens axes. Here the 'cameras' correspond the user's eyes in the virtual world and the offset between them should also correspond to the users eye separation.

3D application programmer interfaces (API) introduce a concept called view frustum. Geometrically it is a truncated pyramid, defining the volume considered when rendering an image of a virtual scene to be presented on a display surface (see Figure 3.1). The volume is limited

also by the near and the far clipping planes. The objects inside this volume will be visible, while the others are left out. The apex of the pyramid is in the camera, or in the eye of the user, and the pyramid is expanding towards the display surface, usually even beyond it. The shape of the frustum also defines the aspect ratio and the vanishing point of the rendered image.

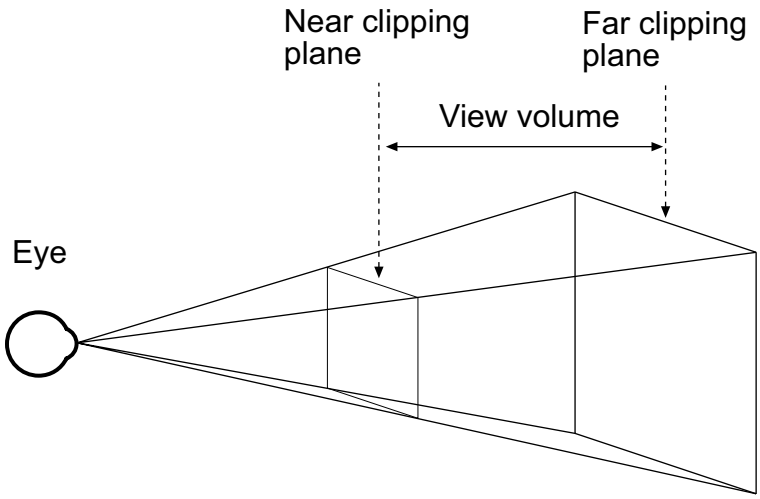


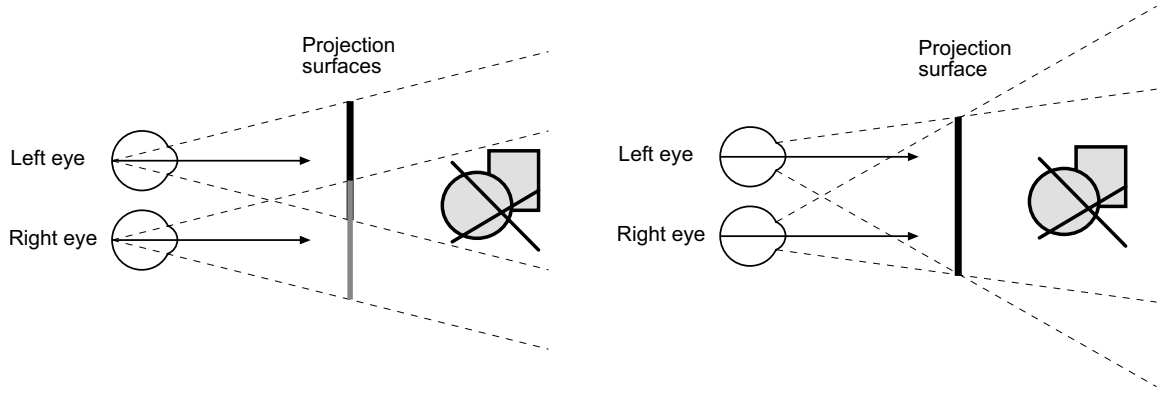

Figure 3.1: Geometric definition of view frustum.

Figure 3.2 (a) presents the viewing geometry in 2D in a case where both eyes have their own displays (HMD). Figure 3.2 (b) illustrates asymmetric view frustums for time sharing techniques where the same display surface is used for both eyes. In a case of HMD (a) the eye is located all the time in the middle of the display and therefore the frustum is symmetric. In other words, the angles between the direction of view (lens axis) and the lines between eye and the edge of the display surfaces do not differ. The other case (b), where the eye is not in the middle of the display surface but is closer to one or the other edge, the previously mentioned angles differ. The difference is bigger, the closer the eye moves towards the edge. To provide a geometrically proper image for the user, the shape of the defined frustum must meet the real physical situation. This is achieved by parameterizations of the view frustum. The shape of the frustum can be set statically, or changed dynamically to correspond to the location of the user while running an application.

Once the system generates the projections correctly, they must be shown to the user with a frequency high enough to avoid flickering. Using shutter glass technique, 60 Hz per eye (120 Hz total) is considered as an adequate frequency for high-end VR facilities.

**Field of vision**

Wide projection, filling the user's field of vision as completely as possible, is also an important feature in a VR system. When the computer generated scene fills the user's field of




(a) Symmetric viewing geometry for two separate displays.

(b) Asymmetric viewing geometry for time sharing visual systems using one display.

Figure 3.2: Symmetric and asymmetric projection geometries for different purposes.

vision entirely, the feeling of true presence is attainable. If the field of view (FOV) is narrow, the user feels more like using binoculars to observe the virtual scene than really being *in* the scene. FOV refers to the displayable area measured in degrees. The use of narrow projections causes a need to darken the rest of the user's field of vision. If the user is able to see also the real environment on the edges of his field of vision, a strong immersion is not achieved.

It is very effective to implement a wide field of vision by using multiple large, synchronized display surfaces placed around the user. These kind of systems are often referred as SIDs, virtual rooms or alternatively as caves after the original Cave Automated Virtual Environment (CAVE) installation [Cruz-Neira et al. 1993].

### Quality of the picture

Computer generated projections of the virtual world are presented as raster images with a certain resolution. By increasing the image resolution, it is possible to decrease the amount of artifacts caused by the quantized representation. The image provided to the user will also be smoother. The quality of the picture can be enhanced by using a technique referred as anti-aliasing. It aims to fade out individual pixels to produce smoother image. On high-end VR facilities, real-time anti-aliasing is always used to avoid confusing artifacts on still images and disturbing flickering on moving scenes. If the user must pay unnecessary attention to low quality presentation of the virtual scene, the immersion is again unlikely strong enough for a credible experience.

**Dynamic projection geometry**

Dynamic projection geometry allows the user to move himself in the real physical space and observe the changes in the virtual scene caused by his own movement. The user may peek behind virtual objects just by moving his head in a natural way. In a case of a virtual room like the EVE, the dynamics keep the view natural even the user moves back and forth between the display surfaces. System without this feature does not respond naturally to users behavior and hence does not support the experience of being inside the virtual world and participating in the existence of a virtual environment.

To provide these dynamics, the view frustums and camera positions in the virtual world must be changed continuously according to the real movement of the user. In a case of multiple display surfaces, there must be different view frustum for each display. The frustums may be also asymmetric, which is usually the case in virtual rooms. For example in a four wall installation, four (monocular case) or eight (stereo imaging) view frustums are defined, one or two for each display surface.

Figure 3.3 presents monocular situations from bird's eye view, related to two different user positions in a virtual room facility. In a case of stereo imaging, the frustum must be defined separately for both eyes. In case 1), the user is located in the middle of the display surfaces and all four frustums are symmetric and equal to each other. In case 2), the user has moved a little forward and near the right wall. This has caused the frustums to change asymmetric and disparate.

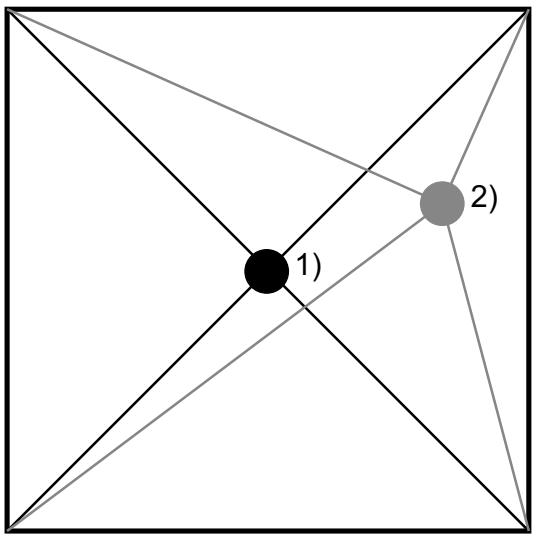



Figure 3.3: Monocular frustums for four surrounding displays of two user positions seen from above.

### Audio

It is said that in a movie the audience gets 70 % of the information from the picture and 30 % from the sound track, while 70 % of the atmosphere is created by the audio and only minor 30 % comes through visual stimuli [Seiro 1998]. In a virtual reality the role of audio is also very significant, specially in applications focused on creating comprehensive and immersive environments. Adding suitable audio in a silent model will immediately increase the credibility of the experience. Audio systems may utilize either headphones or several loudspeakers. There are installations (e.g. Tampere VRC) that use very wide dynamics to product ultra low frequencies with acoustic pressure, so that the user more like feels than hears the sound[1].

Virtual reality also provides a possibility to create artificial virtual acoustics and thus auralize[2] sound sources realistically in a given space. For example in the design phase of a concert hall, virtual acoustics will give the designer concrete feedback and reveal the acoustical problems before entering the construction phase. With this application area the planning work of different spaces may benefit directly from the VR techniques. High-end virtual acoustics are discussed more in [Savioja 1999].

### Input devices

A comfortable and effortless way to interact with the virtual world is an important part of the immersion. If the user interface is too cumbersome, the actual task is left for less attention because the user is burdened with managing the basic navigation commands. The traditional keyboard and mouse are not suitable for true 3D environments, and it is obvious that special input devices are needed to implement a proper VR user interface.

In an immersive virtual environment, different tracking devices are very useful and are also often used as elements of larger systems. Tracked wands and data gloves are nowadays common instruments in virtual rooms [Laakso 2001]. Speech recognition is also taking steps to be a useful interaction technique. In the future, different kinds of touch pads and PDAs' may be adopted as VR input devices. The trend is to use a combination of many different input techniques in comprehensive VR applications.

[1] <URL:http://www.vrc.tut.fi/audio.html> 17.5.2001

[2] Auralization refers to a signal processing chain where source, medium, and receiver are modeled separately. The sound reproduction utilizes the features of all components to produce a realistic soundscape [Savioja 1999].

#### Haptic feedback devices

Force, tactile and kinesthetic motion feedback techniques can be used to enhance a multisensory experience. Often these devices are integrated together with actual input devices. For example, data gloves may be extended with tactile feedback elements attached to fingertips. Different dynamic skeleton systems are used to limit the movements of a users body according to the obstacles in virtual world. Simulating walking on a terrain in a virtual environment needs specific facilities physically manipulating the shape of the floor under the user [Äyräväinen 2000], [Iwata et al. 2001].

Most of the existing haptic techniques are in an immature stage. The basic ideas usually work on the drawing board but the actual equipment is not sophisticated enough to provide credible physical feedback from the virtual world. With the present haptic devices we can just get glimpses of techniques actually usable in the future.

### 3.1.1 VR display types

The mostly used virtual reality display equipment can be divided in three different categories [Jalkanen 2000]. Not only the operational principles differ, but also the purpose of use and the amount of possible participants in the virtual experience varies. The next subsections introduce these three different approaches for implementing the visual part of a virtual world. While the two others are intended to create an immersive experience, the VMD is used to illustrate 3D models in an aquarium style. It is discussed here because the technology is very similar to the spatially immersive displays. For a reference, another division custom is presented in [Wright 2000].

#### HMD - Head mounted display

A head mounted display (HMD) is usually a helmet-like personal display device, which has two display elements, one for each eye (see Figure 3.4 (a)). The display elements may be implemented using cathode ray tube (CRT) or liquid crystal display (LCD) techniques. Independently controllable displays are able to present different pictures for the left and the right eye to achieve stereo imaging introduced in 3.1. A CRT-based HMD is the oldest display type used in VR genre. HMDs may be equipped with head tracking to provide the user a natural way to move his view point in a virtual world.

It is possible to build HMDs using relatively inexpensive components, which is why the equipment is widely spread. However, the mid-price, low resolution (e.g. 640x480) HMDs can not be considered as high-end VR technology. The most expensive HMDs costing up

to tens of thousands US dollars, even beyond that[3], are more applicable to professional VR applications with larger resolutions (e.g. 1280x1024), but still suffer from the basic problems related to the HMD technique. Low resolution, narrow field of view, heavy weight causing physical annoyance, latency problems, and lack of multiple simultaneous users, are topics describing the problems. An important issue is also that the user does not see his own body while using a HMD. Therefore, there is usually a need to create an avatar to represent the user in the virtual world. Without the visual awareness of one's own body, the immersion and the feeling of really being a part of the virtual scene does not come true. A few of the VR input devices are also hard to use without a natural visual contact.

There are positive features that encourage the usage of low-cost HMDs. They are economical, moveable and they do not require too much space or installation work on a site where they are intended to be used. HMDs are also the way to bring VR from research laboratories to masses.

(a) Head mounted display from Honeywell International Inc.

(b) Holobench from TAN GmbH.

Figure 3.4: Examples of a head mounted display (a) and a virtual model display (b).

### VMD - Virtual model display

After maturing enough, the CRT technology reached refresh rates needed to accomplish stereo imaging with a single monitor and shutter glasses. The concept was also extended for larger displays. The idea of a VMD is to present 3D objects near the user in a stereoscopic

[3]A HMD survey is available from <URL:http://www.mmc.tudelft.nl/education/hmd-case/> 2.6.2001

'window' or on a 'desk', or on a seamless combination of them. Typically the displays use back-projection technology to avoid the shadow of the user. VMDs' displays are usually 2-3 meters wide and 1.5-2.5 meters high. An example, Holobench from TAN GmbH, is presented in Figure 3.4 (b). The VMD systems may also utilize head tracking to provide an easy way to change the viewpoint when examining for example computer aided design (CAD) models or scientific visualizations. Other VR input devices may as well be used to make the controlling of the applications more convenient.

VMD is characteristically like a big screen Therefore, VMDs are used mainly with applications that present models in front of the user. A deep immersion is not achieved because of the user's awareness of the presence of the display equipment itself. A VMD can not create a surrounding environment. Like the HMDs, either VMDs do not require big special rooms to fit in, and they can be used more widely than large scale SIDs. Though, VMDs are still large enough to extend the personal use of a desktop monitor to a multiuser visualization tool.

**SID - Spatially immersive display**

The basic idea in SIDs is to surround the user with a stereoscopic picture. With this technique, the immersion becomes so strong that the user does not consider the situation as looking at a screen from outside, but more like being inside something. Technically SID is like a multiplicated and scaled VMD, but the conceptual difference is that the user will always see a VMD as a big monitor and understand that the presented material is actually attached to the display equipment. In a optimal case of SID, the user forgets the screens and equipment and all attention is paid to the three-dimensional content of the scene. With SID, the virtual world becomes a part of the users reality.

SIDs are divided into curved wall setups (see Figure 3.5 (a)) and CAVE-type setups (see Figure 3.5 (b)). Curved wall technology is used mostly in show rooms and it is intended for bigger audience while the latter is suitable for personal use and collaboration of small groups. The original cube-shaped facility built at Electronic Visualization Laboratory[4] at the University of Illinois was called CAVE [Cruz-Neira et al. 1993]. The name, CAVE, was soon adopted as a common noun for these kinds of setups. Today CAVE is a trademark used by FakeSpace Systems[5] and the company sells a commercialized version of the CAVE, which is based on the original design.

This thesis concentrates on virtual environments created by using CAVE-type SID facility. Typically they are cube-shaped frames using three to six back projected display surfaces. For example, three wall installation provides 270 degrees field of vision. The user can move inside the cube and examine the virtual world rendered as stereoscopic images on the sur-

[4] <URL:http://www.evl.uic.edu/index.html> 25.5.2001
[5] <URL:http://www.fakespacesystems.com/> 25.5.2001

(a) Curved wall display, Reality Center from Silicon Graphics, Inc.

(b) Virtual room from Silicon Graphics, Inc.

Figure 3.5: Examples of two different SIDs from Silicon Graphics, Inc.

rounding screens. Genlocked image signals are usually generated with a state of the art SGI's graphics hardware and projected with CRT video projectors. High refresh frequencies (100-120 Hz) required for high quality stereoscopic shutter glass images can be driven only with CRT projectors with a fast green phosphor. Therefore, unfortunately cheaper and also brighter LCD projectors can not be used [Napari 1999a]. High refresh rates and very fast decaying phosphors are necessary to be able to produce sharp and clean images between every swap from the projection of one eye to another.

Comparing the three different approaches in creation of a visual virtual reality experience, SID is the most expensive and the most complicated system. It consists of many valuable components, it demands also a lot of room and it is definitely the hardest to move from one site to another. Still, it has many benefits compared to the other options. It provides a virtual experience for multiple simultaneous users, it does not demand the users to wear heavy gear and it is not vulnerable to the delay related to changes of the user's head orientation. The last issue is a tremendous benefit compared to the HMD technology. When a HMD user turns his head even slightly, the projection must be changed immediately. However, the user can easily make such a rapid movements that the rest of the system can not follow the changes fast enough. To the user, this appears as an inconsistent image and an inconvenient delay in projection update. In a SID, the projections are pre-calculated for very large field of vision and presented all the time on the surrounding screens. Therefore, it does not suffer from the effect caused by the delay, and it allows longer periods of continuous use without a simulator sickness. In a SID, the user stays aware of his own body and sees for example the input devices and his own hands naturally, not as clumsy avatars. This feature increases the

convenience and makes the use of the input devices easier.

Next section introduces the SID-type EVE facility at Helsinki University of Technology. The EVE was used as the implementation environment of the applied part of this thesis.

## 3.2 EVE - Experimental Virtual Environment

The VR infrastructure in Telecommunications Software and Multimedia Laboratory at HUT began to evolve in 1997. At that time, the initial funding was arranged for the design and construction of a CAVE-type SID facility. In the first phase, a project called HUTCAVE, carried out the construction of one wall VR setup, capable of utilizing stereo imaging and magnetic tracking. Audio features were also integrated on the system then. The second phase of the project moved the facility to a new Computer Science building and extended it as a four wall immersive projection system including the front wall, the side walls and the floor. Figure 3.6 illustrates the first constructional design of the second phase setup. Later, the layout of the video projectors has been changed. Also the name HUTCAVE was changed to the EVE – Experimental Virtual Environment, because of the acronym 'CAVE' was already trademarked by FakeSpace Systems. The whole development project from the beginning until the stage where the frame and canvases for four display surfaces were completed, is fully documented in [Jalkanen 2000]. At that time, the rest of the hardware supported only the front wall projection.

At the time of writing this, the second phase is still ongoing, but the construction is just lacking of finalizing the floor projection, and completing the audio system installation. The present hardware configuration can be divided in five subsystems introduced below.

#### Projection system

The visually dominant part of the virtual room consists of an aluminum frame and display surfaces. Together they form a cube shaped construction. The user stands inside the open cube, whose each side is approximately 3 meters long. The front wall and the sidewalls are back projection canvases made of Harkness Hall[6] Polarcoat material. In the near future, the floor will also be used as a projection surface. In that case, the picture will be front projected from the ceiling to the painted floor surface.

The images for each display surface are projected with Electrohome Marquee 8500 LC Ultra CRT-type video projectors. For the user, who is standing in the middle of the virtual room, the present configuration provides 270 degrees field of view, which is enough to create a strong immersion. The nine square meters floor provides enough space for about five

[6] <URL:http://www.harknesshall.com/> 25.5.2001

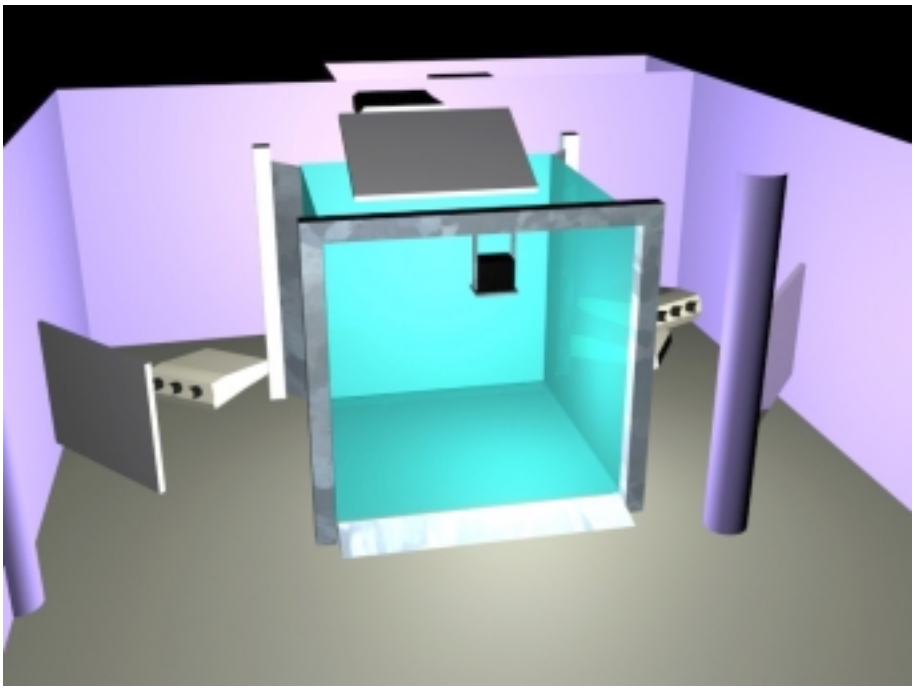


Figure 3.6: Original design of the second phase of the EVE.

simultaneous observers.

### Graphics system

The core part of the visualization system is a SGI Onyx2 computer with two Infinite Reality 2 (IR2) graphics pipelines. The computation is powered with eight 250 MHz R10000 MIPS processors and 2 GB of RAM memory. The graphics pipelines provide an additional 64 MB of fast texture memory. By the experience gathered in many projects in the laboratory, it can be said that the system is capable of running smoothly the 3D models size of under hundred thousand triangles and the amount of textures not reaching the size of dedicated texture memory. The real-time graphics computing power in the pipelines is based on a hardware implementation of the OpenGL (see Section 4.3.2) graphics standard.

The special features that still keep the SGI hardware ahead from the PC technology, are the real-time antialiasing for big resolutions, such as 1024x1024 pixels, and the capability to synchronize multiple display surfaces to provide consistent hardware level image drawing on every frame on every display. The latter is required for example to be able to create a stereo effect on a multipipe display system.

### Stereo images

The stereo imaging is implemented by using quad buffering and Crystal Eyes shutter glasses from StereoGraphics Corporation[7]. Quad buffering is supported by OpenGL graphics standard. It is an arrangement to have different front and back buffers for both eyes. The benefit is that the images for both eyes can be swapped from the back buffers to the visible front buffers at the same time, and the only delay between the eyes comes from the refresh frequency of the particular graphics format. The EVE is typically driven with refresh rate of 120 Hz, hence the delay is about 8.3 milliseconds.

The synchronization between the images for the left and the right eyes, is implemented by using infrared signal from multiple emitter units. The emitters are placed on the corners of the aluminum frame of the EVE and a sensor on the glasses receives the synchronization pulse.

### Input and interaction devices

For a long time, the magnetic tracker and the wireless mouse have been the main input devices in the EVE. Utilization of the data gloves from 5DT[8] and speech recognition techniques from IBM[9], have been recently under development together with the successful attempts to apply tracking and wireless mouse in new ways [Laakso 2001].

The tracker device is Ascension[10] MotionStar with six 6-degrees of freedom (DOF) sensor units. The device is controlled by an industrial PC and the position and the orientation information is transferred over Ethernet local area network (LAN) to the main computer, which runs the applications. The tracking device is used to track various motions. It has been used for example in the user head tracking, the navigation wand tracking and the orchestra conductor following. For instance, information about the user's head position and orientation is used to dynamically optimize the view projections according to the movement of the user in the EVE.

### Audio

Since the goal is to create comprehensive virtual experiences, the audio is also very important. The EVE is equipped with an exceptional virtual acoustic system designed by the DIVA research group[11] from TML at HUT. The system consists of fifteen Genelec 1029 active

[7] <URL:http://www.stereographics.com/> 17.5.2001
[8] <URL:http://www.5dt.com/> 17.5.2001
[9] <URL:http://www-4.ibm.com/software/speech/enterprise/te_3.html> 17.5.2001
[10] <URL:http://www.ascension-tech.com/> 17.5.2001
[11] <URL:http://www.tml.hut.fi/Research/DIVA/> 17.5.2001

monitoring loudspeakers placed around the visual projection system and a subwoofer. The loudspeakers are driven by an appropriate audio hardware and software.

In the EVE, a dual-processor PC, which runs Linux operating system, acts as a sound processing unit and is used for the room acoustic modeling and auralization, sound source panning, and equalization filtering [Hiipakka et al. 2001]. The PC is equipped with RME's Digi9652 audio interface card with ALSA[12] drivers. The signal processing software has been written at TML and is divided into two parts. It consists of a low-level library DIVADSP, and a flexible high-level signal processing tool, Mustajuuri[13].

To provide good acoustical conditions, the walls and the ceiling have been mainly covered with an absorbent material to avoid echoing. In the future the unnecessary noise sources will also be arranged to be more silent. This includes, for example, changes to the air conditioning system.

[12] <URL:http://www.alsa-project.org/> 17.5.2001
[13] <URL:http://www.tml.hut.fi/~tilmonen/mustajuuri/> 17.5.2001

# Chapter 4

# Implementation

## 4.1 Requirements for virtual reality application in the BS-CAVE project

Virtual reality applications are real-time systems by their nature. The requirements for real-time visualization systems are discussed in general in [Earnshaw and Jern 1994] and [Bryson 1994] while the focus in [Bierbaum 2000] is especially in VR related real-time issues. The primary target is to raise the overall performance of the system on such a high level that none of the features of the system causes annoying emotions to the observer. Rising the performance can be understood also as minimizing the delay between the user action and the system response. Part of the requirements must be taken into account in the low level VR software framework, which lies under the actual application, while the rest of them must be managed on the application side. An appropriate framework software will solve and hide the low level problems from the application programmer. They are related for example to the multiprocessing, message passing, shared memory, dynamic memory management, synchronization of different processes, and fixed display frame rate. In other words, the implementation of the framework must use the features of the underlying operating system and hardware as efficient as possible, but still keep the architecture flexible for future extensions and improvements.

The framework should encapsulate all of the input device drivers in a way that the programmer just needs to ask for the current values without having to worry about different threads, processes and synchronization issues. If the VR framework does what it is supposed to do, it provides a convenient and efficient environment for application development and operation. After this assumption, it is on the application programmer's responsibility to use the framework in a way that the performance remains acceptable. This section discusses the issues that are left for the application developer.

If it is assumed that there is enough computation and graphical resources in the system for creating a high quality virtual environment, it is up to the application designer not to overload the system. The application should not try to do anything that simply is impossible with the current system configuration. In the other hand, the application must use the resources efficiently enough to achieve at least the minimum performance level required. The better the frame rate and the faster the system responses are, the better and more immersive the virtual experience is to the observer.

The overall amount of data in an application must not reach the system limits in a way that the data should be swapped continuously back and forth between the disk and the memory. The disk operations make the performance easily very poor. The complexity of the 3D-model, the polygon count and the usage of the texture memory, must also stay inside certain limits to keep the frame rate above the required convenience level.

In a practical level in this project, the requirements for the visualization system can be defined as follows. It is assumed that there is a proper framework available. The stereo imaging, the user tracking and the dynamic frustums are supposed to be part of the base functionality of the framework. The following requirements have effect on the application and the size of the models.

- The user can choose by interactive manners, which kind of visualization elements are shown.
- The visualization should take advantage of the true 3D environment.
- The user can move freely in the virtual space.
- The scene is shown in a convenient frame rate.

The first requirement indicates that there should be alternatives on how to present the behavior of the flow. The user should control also what is seen and when. Interactivity refers to the feature that when a choice between the visualization alternatives is done, the system responses immediately.

The second issue means that the methods used for visualizing the invisible air flows should be implemented in a way that they bring additional benefits to the resulting perception compared for example to printed pictures. The environment is truly three-dimensional and it should be brought out also on the essence of the visual elements, which are used to present the numerical flow information.

The third requirement about free movement is a critical matter. The idea of a virtual world that represents something actually nonexistent and makes the observer to feel it to be real, needs the same freedoms as the physical and real world would allow. A freely navigable

model is one of the preconditions of proper immersion. In the BS-CAVE project, one objective is to improve the navigation techniques used at the EVE. The resulting methods will be combined with the flow visualization features developed during the project.

To fulfill the fourth requirement of the convenient frame rate, or in other words, the high enough overall performance, compromises are still always needed. The convenience of the frame rate is an individual question, but an average limiting value is about ten frames per second (fps) [Bryson 1994]. If the application tries to use too much of the memory, a lot of time is spent in swapping the data. In the worst case, swapping is needed between every rendered frame and the performance will go down. Depending on the power of the graphics subsystem, there is always a limit for the complexity of the model. A model more complex than this limit allows, will cut down the frame rate and the high-end performance is again doomed. The model is too complex if the amount of the polygons inside the view frustum exceeds the limit of processing capability or the dedicated texture memory is not large enough for the materials in the 3D-model. The user is responsible for avoiding these conditions.

The frame rate may drop also in a case of too heavy computation. If the objective is a convenient frame rate, the performance is not acceptable if the simulated virtual world in the application requires more processor time for the computation than it is available between the frames. The application programmer must always be aware that the resources available in a real-time system are limited, and the limit will be reached often too easily.

## 4.2 Potential software components

### 4.2.1 Importance of right tools

The software is the one that tells the hardware to do whatever it is meant to do. If the software do not use the hardware properly, the money spent on the machines and the displays is wasted. The system is just as powerful than is its weakest component. The hardware is useless without a proper software, and vice versa. That is why the selection and the implementation of the software components are as important as is the evaluation of the hardware before purchasing it. It can be very slow and frustrating to develop the software from scratch. Even if the developer had a bunch of tools and earlier implemented libraries were available, the work might progress slowly. It is very important to choose the right tools and the environment in the first place. It may be extremely depressing, if after a few months of work it is noticed that the tools are not applicable enough for that specific project. Wrong tools do not just waste the valuable motivation, but also money and time. The resources in this project did not allow to write the whole implementation from scratch, besides it is almost never wise at all. It was essential to find the suitable building blocks. For these reasons, consider-

ably amount of time was spent on evaluating the potential software components and finally choosing a set of them to work with and to implement the visualization software package.

### 4.2.2 DIVA and LibR

"Which generic VR software platform should be used in the EVE?" This was long time an open question, in fact the question existed even long before the BS-CAVE project was started. In the beginning of the VR research and development in Telecommunications Software and Multimedia Laboratory at HUT the first bigger project besides the physical construction of the virtual room itself, was the virtual orchestra called DIVA. DIVA consists of a 3D model of a virtual concert hall and animated musicians, a virtual acoustic solution and an ability for the user to conduct the orchestra with a tracked baton and a glove. LibR is the software platform that was developed for DIVA project, and it became the first actual VR framework used in the EVE [Hänninen 1999]. It is a complex C++ library, which includes animation control, kinematic modeling and real-time 3D graphics rendering, which supports both the monoscopic and the basic stereo imaging. It is very much DIVA oriented and extending it to be used as a generic platform would demand a lot of changes and rewriting of the code. The strengths of the LibR are the unrestricted extendibility and the techniques for inverse kinematics and auralization. The two latter features are used separately also in other research projects. The extendibility is actually a theoretical issue because of the strong monolithic structure of the implementation. However, LibR was evaluated to need too much effort in a case it would be taken in use as the generic the EVE software platform. The current implementation does not support multi thread rendering, culling, or level of details. The input device support is also very limited. Either the lack of tutorials or programmers guide and required lots of experience in C++, OpenGL, X-Window system and tool command language (TCL), do not make the utilization easier. The idea to use LibR was little by little left on the background.

### 4.2.3 Getting closer - Avango

In the academic world, one boundary condition is that it is hard to get funding on anything that costs more than nothing. That was why the commercially available but expensive software systems were out of the question right from the beginning. With a moderate price, there is not much to choose from. The staff from the faculty visited in a few locations in Europe to get information about the other software solutions in the high-end VR applications. It was very promising to have a possibility to use a software called AVANGO[1], which is developed in the GMD research institute in Germany. The EVE would had been a beta

[1]Formerly known as AVOCADO, name was changed due to trademark a conflict.

tester for the development version of AVANGO. AVANGO is based on IRIS Performer (see Section 4.3.2) and the implementation is a combination of C++ and Scheme. AVANGO is focused on the framework development for distributed VR applications [Tramberend 1999]. It uses the features of IRIS Performer and adds its own nodes for example for audio. In spite we got a release of AVANGO to be tested in our laboratory, we were not able to take it in serious use mostly because of the lack of the documentation and the support from the original developers.

### 4.2.4 Evaluating several alternatives

While the hunt for a proper software framework was going on, the EVE still needed a software for the demonstrations of the first phase of the virtual room setup. The setup had one functional display surface. The potential sponsors and project partners had to be convinced that the development of the facility would be worth an effort. To fulfill this need, a software called Stereofly was developed [Napari 1999a]. It is based on the Perfly demonstration software distributed with IRIS Performer. Perfly provides an ability to load and present any 3D model whose format is supported by the set of IRIS Performer loaders. In addition to Perfly's original features, Stereofly adds a high-quality stereo imaging with a support for Ascension MotionStar tracker. For the first time, it was possible to explore a wide range of 3D models from different view points only by moving around in the virtual room. At that time only one wall was operational, but it really was an encouraging start and the laboratory got an another amusement tool for the visitors, besides DIVA. Although Stereofly provides a way to ride through the models, it lacks a proper support for additional input devices, audio, and environment configuration methods. It is also so called "quick hack" and it has properties that are characteristic to the software of that category. Therefore Stereofly is not suitable to be taken as a basis for development of a whole VR software framework. Anyway, it is still a good tool for the quick demonstrations.

In the summer 1999, TML had resources for a broader survey of the potential VR frameworks and toolkits. On the background of the survey there was a publication "Software Tools for Virtual Reality Application Development" from Bierbaum and Just [Bierbaum 1998]. The work at TML produced an extension for the publication referred above. Three more software packages were evaluated in the survey [Laakso 1999]. The software packages that were not available on the SGI platform were rejected right in the beginning. LibR and IRIS Performer were already well known. CAVELib[2], dVISE, and World Toolkit[3] were out of the question at that time because of the financial issues. Furthermore, MR Toolkit[4] was already evaluated

[2]<URL:http://www.vrco.com/products/cavelib_main.html> 24.5.2001
[3]<URL:http://www.sense8.com/products/index.html> 24.5.2001
[4]<URL:http://www.cs.ualberta.ca/~graphics/MRToolkit.html> 25.5.2001

not to solve the framework problem. At that time, Lightning[5] was left to wait for further development. Thus the survey concentrated on Maverik[6], Open Inventor[7] and VR Juggler (see Section 4.3.4).

### Open Inventor

Open Inventor was known to lack many features needed to develop a mature VR framework, but we still wanted to test a few ideas with it. As a result of the work, a VR viewer software, Ciview, was developed. The basic idea is to move a bunch of cameras in a 3D model and render several images to be projected on the display surfaces of the virtual room. Ciview supports the user head tracking and stereo imaging. The stereo effect is created by using shutter glass technique with dynamic and asymmetric view frustums. The dynamics is based on the tracker information. Ciview is built mostly on Open Inventor while the stereo rendering utilizes directly the OpenGL stereo buffer features. In the first version, the images are rendered into different windows according to the configuration information. One limitation is that the only supported model file format is Open Inventor. Major problems are related to the limited performance originating from the lack of a multiprocessor support. The strengths of Open Inventor are ease of use, extensibility, and readily available interaction methods, but as a single thread system it was not suitable for the basis of a high-end virtual reality software.

### Maverik

Maverik was also investigated deeply. The evaluated version 5.0 was released in May 1999 but the development of the software has lately reached a version number of 5.4. The work is still active. The architecture of Maverik is based on a microkernel and a collection of modules providing display management, interaction, control of input devices etc. It is available for the all usual platforms. The package offers a complete programming guide, tutorials and sample source code. Therefore, it is easy to be taken in use. During the evaluation, a small prototype of a viewer for a virtual room was implemented in a fair short time. The version 5.0 was implemented for stand alone systems only, but since then a complementary system called Deva has been developed. It provides a distributed environment for the networked users. In spite of the advantageous features, also Maverik suffered from weaknesses. The main reasons, why it was not suitable for the EVE, were the lack of support for the high-quality stereo imaging and for the Ascension MotionStar tracker device. The overall performance was not as good as was required. With the simple models, Maverik performed

[5] <URL:http://vr.iao.fhg.de/lightning/index.en.html> 25.5.2001
[6] <URL:http://aig.cs.man.ac.uk/maverik/download.htm> 25.5.2001
[7] <URL:http://www.sgi.com/software/inventor/> 25.5.2001

quite well, but with larger models the performance limit was achieved too soon. The rendering was made without an anti-aliasing and it caused the quality of the picture to be too low. At that time Maverik supported only single threads but multiprocessing features were under development. After all, the judgment was that Maverik is an interesting product but too immature to be yet used in the EVE.

#### VR Juggler

The evaluation of VR Juggler resulted an assumption that it would probably be the best choice as the VR framework. Even if the first tested release of VR Juggler, did not support the Ascension MotionStar tracker, but the overall system seemed to be very promising. Later, in the autumn 1999, VR Juggler was studied thoroughly and we were even more convinced that this software had a lot of potentiality to evolve a proper VR framework. The flexible architecture and the configuration system and a possibility to use IRIS Performer as a drawing manager, made the software versatile and being able to perform according to the virtual reality requirements. At that time, we did not have acute projects, which would had required a functional VR framework. Therefore, from our point of view, VR Juggler had time to grow up.

When the version 0.1.95 of VR Juggler was released as an open source (see Section 4.3.1 at page 44) in turn of the year 2000, the direction of the development was still suitable for us. In the winter and the spring, our projects concentrated mainly on the model transfer problems and the file format conversions, but a simple driving simulation was implemented with VR Juggler to test the applicability of the framework. The visualization and the interactivity part of the BS-CAVE project was started in the summer 2000, just before the release 0.1.97 of VR Juggler came out. This release had better documentation and it also had the first version of the MotionStar tracker support. After the first experiments related to the adaptation of new interaction and navigation methods in the EVE, it was clear that VR Juggler would be the development platform in the navigation part of the BS-CAVE project.

### 4.2.5 Solving the puzzle

The situation with the implementation of the flow visualization was different. The first idea in the beginning of the year 2000 was to use software called ELMER Post as the basis for the visualization part of the project. ELMER[8] is a computational tool for a wide range of different application areas in physics, including fluid dynamics. It is developed in CSC - Scientific Computing Ltd. in collaboration with Finnish universities, research laboratories, and industry. The solver uses FEM to solve the problems described as partial differential equa-

[8] <URL:http://www.csc.fi/elmer/> 25.5.2001

tions. ELMER Post is one of the three parts in the ELMER software package and it is used for the visualization of the results from the solver part. It supports many usual visualization elements such as isocontours and isosurfaces, arrow plots, particle traces and cross sections. In the beginning of the project, it was first verified that the flow data obtained from the project partners, was possible to be imported to ELMER Post with a suitable converter. The converter was written and we were able to use the sample data in ELMER Post. Soon after this, there was a considerable setback. After a short investigation of the ELMER Post source code, it revealed that there was no sense at all to try to use ELMER Post as the basis for the development of an application answering to the requirements of the BS-CAVE project. The undocumented code was written by using C and pure OpenGL. In the tests with the software and in the discussions with the original developer, it turned out that the design did not support extensibility and the real-time rendering was never considered in the implementation. Neither the algorithms nor the functional architecture were optimized for performance. There was not much that would had not needed to be re-design and re-written. Among all the other things, the support for the VR input devices and the ability to use photo-realistic 3D models in the same scene with the flow visualization, would had required development from scratch. The only strengths of ELMER Post would had been the visualization algorithms. As clear as is was to use VR Juggler in the navigation development, as clear it was not to use ELMER Post in the BS-CAVE project.

We had to get back to the old drawing board and start the tool selection right from the beginning. At this time, we knew that VR Juggler would be the framework for the other part of the project. On the other hand, VR Juggler offered the possibility to use IRIS Performer as the application building tool, as well as the renderer. This starting point seemed to be very convincing from the project requirements point of view. If we just could use IRIS Performer, importing the 3D models in the application would be a piece of cake. Also combining the visualization solution with the new navigation methods [Laakso 2001] would be easier if the development platform would be the same. The next goal was to find a way to create visualization of the flow data set as IRIS Performer scene graph objects. At that time we did not have the latest knowledge about freely available visualization libraries, but the solution we finally ended up with, now after all seems very obvious.

At that point, we were not sure if there would be suitable tools available for free or for moderate price to complete the chain from computational fluid dynamics solver and 3D modeler to VR environment. After a few days of reading Usenet newsgroups[9] and surfing in the web, the first significant discovery was the conversion library vtkActorToPF (see Section 4.3.3), which led directly to the Visualization Toolkit (VTK) library (see Section 4.3.1). VTK provided the visualization algorithms and vtkActorToPF was the link between the VTK objects and IRIS Performer. In a few brief tests it was confirmed that the conversion from the

[9]For further information see: <URL:http://www.usenet.org/> 3.6.2001

obtained flow data to a format that is supported by VTK, was straightforward to implement and that VTK really provided the required features. In the beginning of the September, the puzzle of tools was finally solved and the actual design and implementation could begin. The tool box consisted of VTK, vtkActorToPF, IRIS Performer, and VR Juggler.

In the next sections the chosen software components are discussed further.

## 4.3 Software tools in implementation

All the major software components chosen to be used in the implementation of the visualization system, are introduced in this section. The evaluation and the selection process is described in the previous section.

### 4.3.1 VTK - Visualization Toolkit

Visualization Toolkit (VTK) is one of the major components of the visualization system implemented in BS-Cave project. VTK is an open-source software product for building data processing, visualization and image processing applications [Schroeder and Martin 99].

VTK is a large and complex object-oriented C++ class library. The native language is C++ but the library includes also an interpreted wrapper layer. It lets the user to use previously compiled classes with Java, Tcl and Python. In this project the applications are built using a C++ compiler.

The basic idea in the use of the VTK classes, is to put together a suitable data processing pipeline to produce the desired visualization as a result. The pipeline begins from acquiring the previously created data from a file, or the data can be generated inside the application in a run-time source class. The acquired data is put through filter classes to extract the usable parts out of the whole data set, to combine different data sets, or to generate more information based on the filter's input (see Figure 4.1).

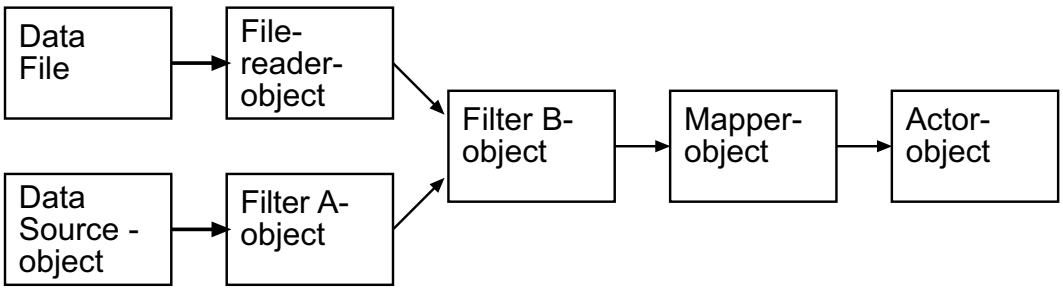



Figure 4.1: VTK Pipeline from data to actor object.

A short and simple VTK-pipeline written in C++ looks like this :

```
vtkUnstructuredGridReader *readerTemperature =
    vtkUnstructuredGridReader::New();
readerTemperature->SetFileName("flowdata.vtk");
readerTemperature->SetScalarsName("Temperature");
readerTemperature->Update();

vtkStreamLine *streamer = vtkStreamLine::New();
streamer->SetInput(readerTemperature->GetOutput());
streamer->SetStartPosition(3.0, 3.0, 2.0);
streamer->Update();

vtkPolyDataMapper *lineMapper = vtkPolyDataMapper::New();
lineMapper->SetInput(streamer->GetOutput());

vtkActor *lineActor = vtkActor::New();
lineActor->SetMapper(lineMapper);
```

The example creates one streamline. The reader object reads the flow data including a vector field and a temperature scalar. The vtkStreamLine class integrates a path through the vector field. The path starts from a specific point. The mapper class is used to map the stream line coordinate values to a graphical object. Finally, the actor object encapsulates the generated object in a renderable structure.

The filters can be implemented in many ways including the separation and the combinations of different pipelines. To import different data sets, VTK has a support for couple of common data formats, besides of its own formats. The supported formats include for example Wavefront .obj, PLOT3D and various image formats. The VTK reader objects provide a broad interface to the results of other software and they enable the use of the best features of various other software tools. An use case of an earlier 2.2 version of VTK on PC platform is reported by P.C.Chen in [Chen 1999].

After the data sets are filtered in a way that the data is ready to be visualized, the output of the filtering part of the pipeline is connected to a suitable mapper class in oder to create graphical objects. Finally, the graphical objects are connected to actor classes to be rendered. Usually a VTK application uses library's own rendering classes to actually draw the picture to the screen, but in this project the rendering is done by IRIS Performer rendering system. The "vtkActorToPF" class that makes it possible, is discussed later in its own section (see Section 4.3.3).

**Open source concept**

Visualization ToolKit is an open source product. This means that the developers of VTK have given the users not only the binaries, but all the source code without a charge. The whole VTK release 3.1.2 including the source code is available from Kitware's VTK website[10]. One of the advantages of the open source concept is that all the users have possibility to make improvements to the software and thus take part to the development work. The users have also right to copy and redistribute the software to make the user community even stronger and to get more potential developers along.

### 4.3.2 IRIS Performer

IRIS Performer [Eckel 97a] is probably the most performance-oriented high-level 3D graphics rendering toolkit for the IRIX and Linux platforms. It is commercially available from SGI. It has been designed to take all advantage of the system resources to achieve the best possible frame rate and real-time performance. IRIS Performer is at its best in visual simulations and virtual reality applications, although it is not primarily designed to be a VR development environment [Bierbaum 1998]. It offers the software developers a powerful programming interface with both ANSI C and C++ bindings including transparent multiprocessing and support for multiple graphics pipes. In the SGI's high-end graphics systems, IRIS Performer is the overwhelming choice as a rendering component. As an almost pure rendering engine, IRIS Performer does not have a support for actual VR input devices. The drivers for the exotic devices must be written by the user or other available driver libraries should be used. Only the mouse and the keyboard are supported by IRIS Performer. Also audio is totally ignored.

OpenGL[11] [Woo 1998] is a low level industry standard graphics library. OpenGL draws points and lines when IRIS Performer, which is built on top of OpenGL, handles larger object oriented entities. The entities are placed as objects in a hierarchical structure called scene graph. Scene graph is a structured description of the objects in the virtual space. The objects are also referred as nodes. The base class of nodes is 'pfNode' and a lot of different nodes are derived from it. In the IRIS Performer scene graph, the nodes know how to render themselves using OpenGL. When the rendering is about to take place, the graph is traversed and each node renders itself. The scene graph structure allows easy scene manipulation compared to a pure OpenGL approach.

An example of IRIS Performer scene graph is presented in Figure 4.2. The node on the highest level, 'pfScene', is used as a root node of the whole scene. 'pfDCS' is a dynamic

[10] <URL:http://www.kitware.com/vtk.html> 25.5.2001
[11] <URL:http://www.opengl.org/> 25.5.2001

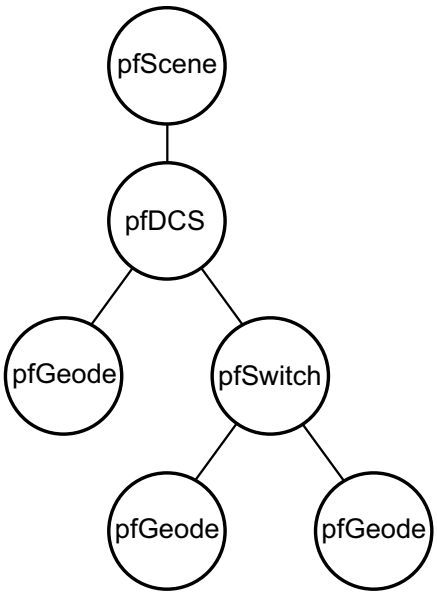



Figure 4.2: Example of IRIS Performer scene graph.

coordinate system and the transformation defined in it has influence to all the nodes below. The ’pfDCS’ has a ’pfGeode’ and a ’pfSwitch’ as its children. The ’pfGeode’ is a container class for geometry sets describing a visual object in the scene. With the ’pfSwitch’ the user can choose, which part of the scene graph under the switch, will be traversed and rendered. In this case, the choice is done between the two different ’pfGeodes’.

Besides the object model and the efficient rendering, IRIS Performer utilizes an optimized and sophisticated memory management. These built-in features reduce the effort from the programmer. In special applications, IRIS Performer manages rendering on multiple channels to drive sufficient amount of data projectors, for example to create synchronized projections around the user in the EVE.

The programming interface consists of six libraries [Eckel 1997b] containing together a huge number of functions (see Figure 4.3). Two of them, ’libpf’ and ’libpr’, are the base libraries for the entire IRIS Performer system. ’Libpr’ stands for performance rendering library and provides a low level high speed rendering. The rendering is based on geometric sets and states defined in the object hierarchy. Above ’libpr’ there is ’libpf’. It is a real-time visual simulation environment with the dynamic scene graph.

’Libpfdu’ provides the programmer with the building tools for the scene and the geometry. The library also contains tools for optimizing the geometry for OpenGL and adjusting the materials and the appearance attributes for fast rendering. To be able to import 3D-models in the commonly used formats, ’libpfdb’ provides a collections of database importers to be used with many popular industry standard formats. IRIS Performer uses internally only its own scene graph memory structure and all the other formats have to be converted before they can

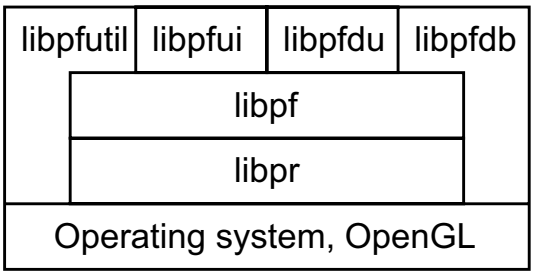



Figure 4.3: IRIS Performer Library Hierarchy [Eckel 1997b].

be used. Most of the importers implement their format specifications correctly, but unfortunately a few of them do not support all of the features of that format. For example, according to the experiences in the EVE, the current virtual reality modeling language (VRML) loader is unable to handle the VRML models properly. The developers can also implement their own database importers.

The 'libpfui' library provides special components for building user interfaces to the applications. It contains many common manipulators and other graphical user interface (GUI) building blocks. To make the IRIS Performer package complete, the utility library, 'libpfutil', adds many convenience routines to the API. For example, the library includes routines for processor isolation, device input, event handling, scene graph traversal, and visual effects simulations.

To get started with IRIS Performer, SGI and the user community, "Friends of Performer"[12], provide a lot of free sample source code, which is distributed together with the IRIS Performer package.

### 4.3.3 vtkActorToPF

The two previous sections have described the features of VTK and IRIS Performer libraries. In the WolfViz software (see Section 4.5 on page 53) that was produced in the BS-CAVE project the former is used to generate the three-dimensional visualization elements and the latter offers the environment for the efficient real-time rendering. VTK does not provide a direct conversion from a VTK object to an IRIS Performer scene graph object. To get these libraries to work together, the programmer needs to make a binding between the VTK objects and the IRIS Performer scene graph. VtkActorToPF is a C++ library written by Paul Rajlich [Rajlich 1998]. As the name indicates, the library is used to translate vtkActors to IRIS Performer 'pfGeode' objects, which describe the geometry and the appearance of the visualization elements.

[12] <URL:http://www.sgi.com/software/performer/partners.html> 24.5.2001

VtkActorToPF is used in the BS-CAVE project to take advantage of both the visualization algorithms of VTK, and the rendering of IRIS Performer. VtkActorToPF can be used in two ways, dynamically to reflect immediately to the changes of VTK pipeline or statically to convert the geometry only when the function is explicitly called. The use of the former method was rejected in the project because of the performance problems caused by the amount of the flow data. In both forms, vtkActorToPF takes the VTK actor object, reads its geometry with the properties and generates 'pfGeoSets' to duplicate the object. The 'pfGeoSets' are collected under a 'pfGeode', which is attached to the IRIS Performer scene graph. The result is equivalent objects both in the VTK pipeline and the IRIS Performer hierarchy.

It is hard to rationalize the option of not to use a third library such as vtkActorToPF. Since VTK is an open source software, it would be possible to modify the source code of the VTK to support IRIS Performer geodes. However, this would require a re-implementation every time there would be a new release of VTK. Use of vtkActorToPF keeps the applications' source code accordant with the paradigm of object oriented programming. Because the problem is solved with additional classes, there is no need to change an already consistent software library.

The source code of vtkActorToPF is freely available from Paul Rajlich's web pages[13].

### 4.3.4 VR Juggler

In virtual reality systems, the software is the element that binds together all the other components. It is the core that gives life for the displays, audio and input devices not forgetting about the computation and simulation. It is obvious that the systems that include many hardware and software components, are very complex. The development of these systems is also a very demanding tasks. In the early years of the VR genre, all the applications had to be implemented from scratch since the software environments were not mature enough to provide the developers with sufficient application building tools. To fulfill the requirements of the virtual reality systems (see Section 4.1), the developers had to have expertise not only in the actual application domain, but also in the very low level programming.

VR Juggler is one of the first attempts to create a comprehensive software platform for the development and the usage of virtual reality applications. It is a very promising open source research and development project founded and lead by the VR Juggler group at Iowa State University[14].

The beginning of the VR Juggler project derives from the need of a flexible framework for virtual environments. The research made in the first phase of the development project

[13] <URL:http://brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF/> 25.5.2001
[14] <URL:http://www.iastate.edu/> 25.5.2001

revealed that none of the existing software systems available at that time, answered to all the requirements [Bierbaum 1998]. The available software fulfilled often only one or part of the demands. To combine these features was hard or even impossible because of the closed architectures or different design approaches. It was also hard to find a system that would had made possible the development of the framework without a need to change and compile the applications after every change in the base hardware system. The development of VR Juggler aims to finally produce a true operating system for the virtual reality.

The current release of VR Juggler is version 1.0.0 and it is available from a dedicated web site[15]. Version 1.0.0 was released after the implementation of the software related to the BS-CAVE project. The version 0.1.97 was used instead. The supported operating systems are IRIX, Linux, Windows NT and Solaris. It provides both a development environment with C++ API and a reconfigurable run-time environment. VR Juggler is an object-oriented modular system with a microkernel core, which acts as a mediator between the applications and the external modules. Most of the components are written in C++ and compiled with GNU compiler[16]. The supporting tools are written in Java.

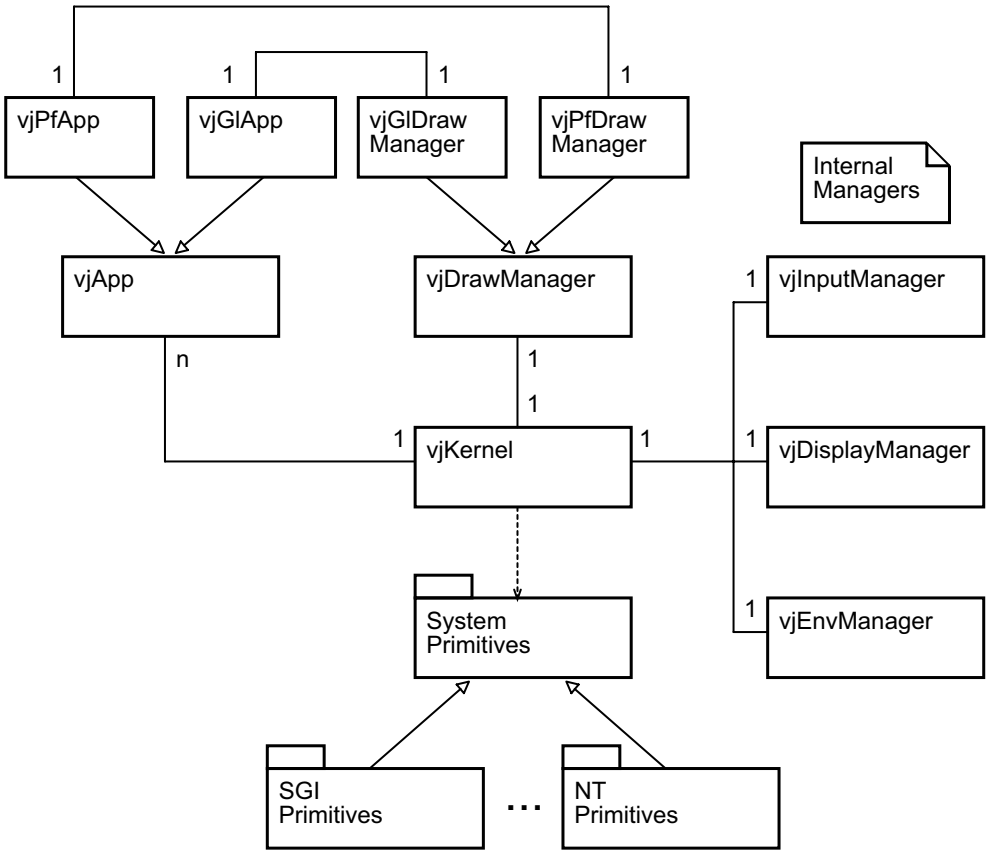



Figure 4.4: VR Juggler microkernel architecture [Bierbaum 2000].

[15] <URL:http://www.vrjuggler.org/> 25.5.2001
[16] <URL:http://www.gnu.org/software/gcc/gcc.html> 3.6.2001

Figure 4.4 presents the VR Juggler microkernel architecture. The designers decided that this conceptual structure would meet the requirements for an extensible, flexible, and robust system. An independent and parallel development work in a group of developers becomes possible in an object-oriented hierarchy, where all interfaces between the components are well defined. The class hierarchy also hides completely the details of the hardware and the drivers. Hence the input devices can be changed at any time without disturbing the running application. The kernel handles the timing for all the other modules and keeps the whole system robust by controlling the processing time given to the processes in separate threads. This way the state of the system is always completely defined and a failure of one process does not bring down the whole system. The users can also thrust that the malfunctions of the system are caused by their applications, not the operating environment.

The kernel uses the underlying operating system features (system primitives) to manage the rest of the VR Juggler system. The internal managers are used for input devices, display settings, managing the configuration information, and for communication with the external applications. In the current release the only external managers are the special draw managers for the user applications. The users can write their own programs based on either OpenGL (vjGlApp) or IRIS Performer (vjPfApp) rendering. The user applications are connected to the run-time system as objects. The interface between the VR Juggler system and the application object depends on the type of the rendering API. The functionalities of the applications are written in member functions defined by the interface.

The microkernel architecture allows the concept of VR Juggler virtual platform (JVP), where the application is on the top of the kernel layer (see Figure 4.5). This model separates the hardware dependent and the hardware-independent components. With its internal and external managers, the kernel encapsulates the system dependencies so that the applications are totally hardware and operating system independent. JVP makes the application development very flexible by allowing the programming to be done on a desk-top computer instead of a high-end VR environment.

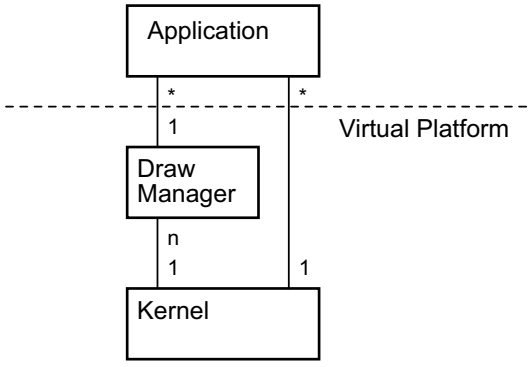



Figure 4.5: VR Juggler application - virtual platform interface [Bierbaum 2000].

The same applications will run anywhere where VR Juggler is installed. If exotic VR input devices are not actually available, their functionalities can be simulated. The applications need to be written only once and they do not need to be especially ported to get them work in other environments. The adaptation to different operating environments is managed with the hardware abstraction and a brilliant configuration system. All the physical elements of the environment are configurable including displays, input devices etc. The configurations are done with a configuration tool, vjControl, which has been implemented with Java. VjControl manages the configurations, which can be manipulated both off-line and while running the applications in the VR Juggler system. For instance, the user can create one configuration, which defines only the display system and another configuration, which defines the input devices available. The configurations are stored in files. Different configuration files can be used together to configure all the system features.

VR Juggler is currently under an active development and the future work aims to develop networking tools for distributed environments, UI libraries and to extend the concept of JVP to a true operating system.

The software written in the BS-CAVE project utilizes the VR Juggler system in the final step of the visualization process (see Section 4.5.4). The rendering and the interaction is implemented on the VR Juggler framework. To be able to directly use the generated scene graph structure, the chosen graphics API is IRIS Performer.

## 4.4 Architectural models

The title of this work refers to a photo-realistic environment. The photo-realism is pursued by a thorough and careful modeling of the architectural spaces. The flow visualization elements will be added in these models in a later phase of the visualization process. The modeling work is done at the Granlund's and the resulting 3D-model files are delivered to the EVE's UNIX environment.

To be able to produce photo-realistic models, the modeler must consider not only the geometry and the materials of the surfaces but also the realistic lighting of the space. Granlund is one of the developers of the VIVA software [Gröhn et al. 2000]. It is a luminaire selection program, which is a remarkable tool for the lighting designers. It provides general information, luminaire drawings, photos, photometric curves and 3D-models about 3000 different luminaires. The information has been gathered from the luminaire manufacturers. With a help from this tool, it is possible to add realistic lighting conditions to the 3D models of different spaces.

One method to achieve very realistic lighting conditions in 3D-models, is the radiosity lighting model. The basic idea of the method is to simulate the radiant energy transfers between

surfaces according to the laws of physics [Hearn and Baker 1997]. In this project the Lightscape[17] software is used for the radiosity computations. The computational accuracy, and therefore the resulting realism in the lighting condition, is proportional to the density of the triangulation of the model surfaces. On the other hand, the great amount of the polygons is bad for the performance of the final VR application. A solution is, that the resulting light maps are combined directly to the textures that are used in the 3D-model. This way the amount of the polygons can be reduced to its minimum after the texture creations. The lighting information is already in the textures and the polygons are needed only to define the shape of the surface.

The modeling process begins from the definition of the geometry with AutoCAD[18], 3DSMax[19] or another proper modeler software. The room models used in the BS-CAVE project are modeled entirely with AutoCAD. After the geometry is defined, it is imported into Lightscape. In that software all the surface properties such as colors, reflectivities and textures are added to the model. The lighting conditions are also defined including the information about distributions of luminous intensities obtained from VIVA. After these tasks, the model is ready for the radiosity computation and the final texture creation.

After the geometry, the materials, and the textures are put together in a modeler software, the model database is ready to be converted and transferred. IRIS Performer is delivered with a lot of 3D-file format importers, or loaders, and a few additional are commercially available. While testing this repertoire it became clear that only a few of the loaders work reliably. The choice for the transfer file format was OpenFlight 14.2 from Multigen[20]. The OpenFlight loader for IRIS Performer is produced by the same company, hence the compatibility of the file format and the loader is guaranteed.

If the IRIS Performer loaders can be problematic, so can be also the file format conversions. Different formats support a different amount of features and there are no guarantees how different converters support different formats. The solution to the conversion problem has been PolyTrans, a high quality 3D model and animation conversion software from Okino Computer Graphics[21]. It can be used both as stand-alone software and a 3DSMax plug-in so that the functionality of PolyTrans is directly available within the 3DSMax. It supports a wide range of different file formats, including Lightscape, 3DSMax and OpenFlight formats. If the edge conditions, mainly related to textures, have been taken into account in the modeling phase, the OpenFlight model file produced by PolyTrans can be used directly with the IRIS Performer based software without further processing. In Figure 4.6 an office room model is successfully transferred to the EVE environment and rendered with IRIS Performer. The

[17] <URL:http://www.lightscape.com/> 25.5.2001
[18] <URL:http://www.autocad.com/> 25.5.2001
[19] <URL:http://www2.discreet.com/products/> 25.5.2001
[20] <URL:http://www.multigen.com/> 25.5.2001
[21] <URL:http://www.okino.com/> 25.5.2001

Figure 4.6: Office room model rendered with IRIS Performer.

effect of the lighting is the most noticeable on the surface of the nearest table under the lamp, behind the monitor, and on the ceiling.

## 4.5 WolfViz - The software family

The software package, which was implemented during the project to complete the flow visualization, is called WolfViz. The name WolfViz comes from words flow visualization. Viz stands for visualization and Wolf is simply the word flow written backwards. The name was chosen because it seemed that all possible names containing flow and visualization in any forms were already used by someone. Wolf is also a descriptive word, not a plain acronym, and it enables a nice mascot for the software.

WolfViz consists of three different modules. Together they form a path from the results of the CFD calculations and the photo-realistic room modeling to the integrated visualization in a virtual environment. The first of these three executables, EnsightToVTK (see Section 4.5.2 on page 62), converts the flow data to a format, which can be read into the VTK pipeline. The second software, Previz (see Section 4.5.3 on page 64), puts the data through visualization algorithms, an object generation, and a conversion of geometry. The resulting visualization elements are stored as an IRIS Performer scene graph in a file in the IRIS Performer binary database format (PFB). In this phase the photo-realistic room model is also stored in the same file. Postviz, the third software (see Section 4.5.4 on page 74), reads the PFB file and actually brings the visualization in the virtual environment. The observer can interact with the model by navigating in the 3D scene and by choosing which of the visualization elements are visible at a time. The components of the WolfViz software package are presented in Figure 4.7 and

each component is discussed in the later sections. The features of Postviz will be combined with the sophisticated navigation methods, which are implemented as the other part of this project [Laakso 2001].

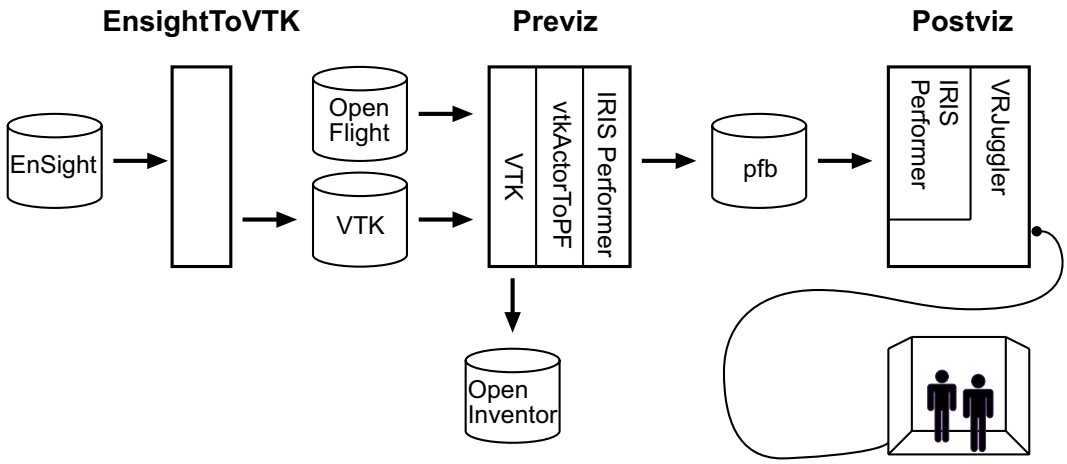



Figure 4.7: Components of the WolfViz software package.

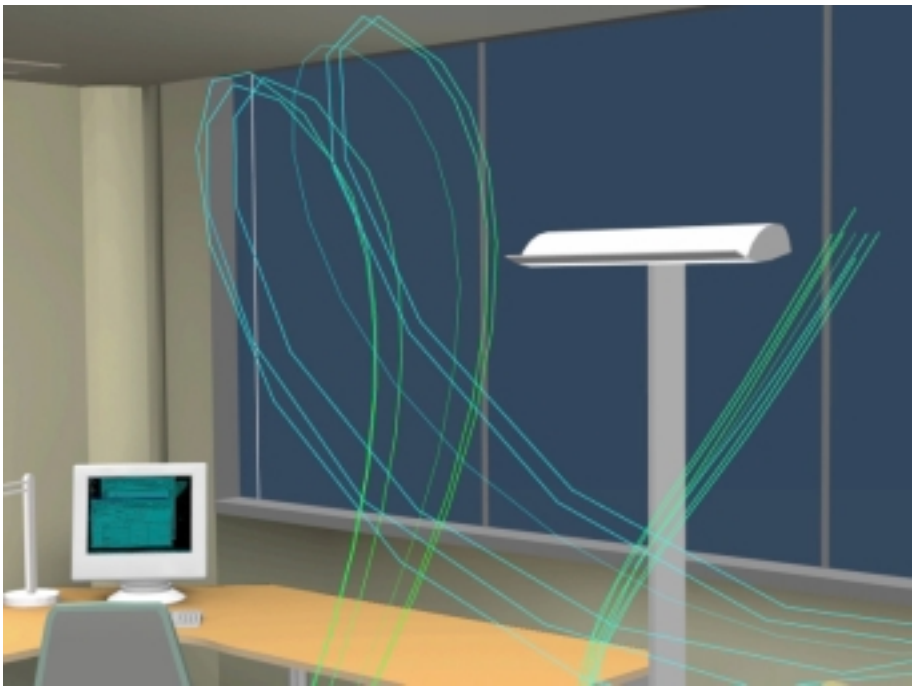


Figure 4.8: Lines illustrating a flow.

### 4.5.1 The chosen visualization techniques

When the visualization techniques are chosen to be used in a real 3D environment, such as the EVE, the criteria differs from the case of an ordinary desktop application. Because the third dimension is really there, the visual elements should take advantage of that feature. An illustrative example of this idea is a comparison of a simple stream line and a stream tube. Even if the line is placed in a 3D space, it is still just a line without a thickness. The observer sees just one pixel wide line, no matter how far the line actually is. In other words, the appearance of the line itself does not give any cues about how far it is from the observer (see Figure 4.8). A tube could be described as an extended line, which have a certain thickness. Contrary to the line, the tube gives a hint about its distance by seeming to be thicker or thinner, depending how far it is. This effect is also called a 'depth cue' [Kalawsky 1993].

Before the visualization techniques were chosen, they had to be evaluated. The alternatives were introduced to the representatives of the project participants as a VR presentation. After a discussion it was agreed, which of the elements will be implemented in the final visualization. During the evaluation it was taken into account that the aim was to create useful visualizations, especially based on the CFD calculations about the ventilation and the heating systems of an office room. The techniques implemented in WolfViz fulfills the requirements of this particular approach. On another application area, the decisions would probably differ from this case.

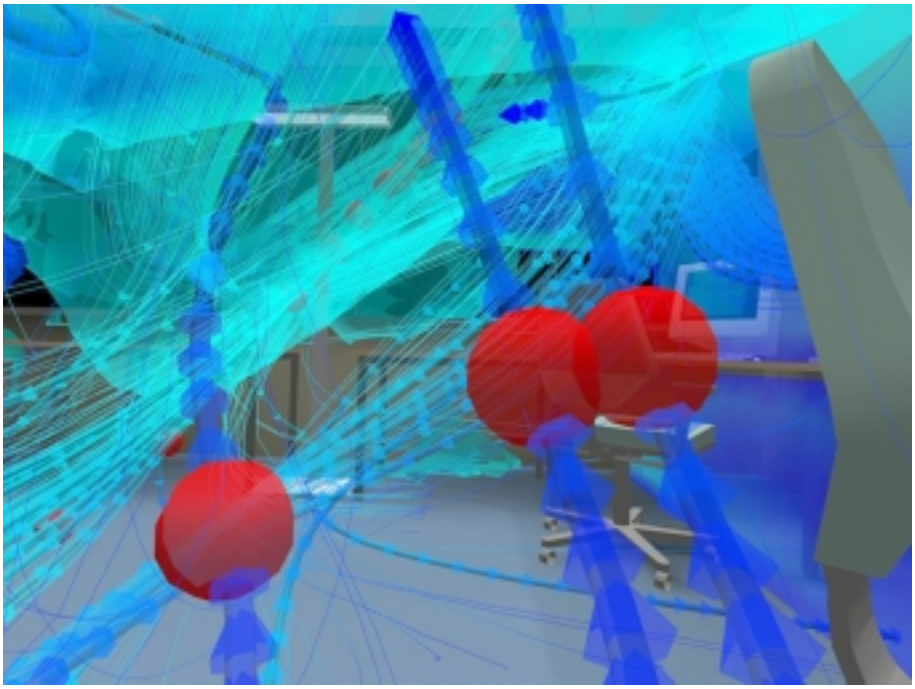


Figure 4.9: Chaotic visualization.

The visualization techniques should not result the elements to lie too close to each other. In

a 3D space, the occlusion can be a problem. Too many visual elements in a small region cause often just a chaotic view. In Figure 4.9 the visualization elements occlude both each other and the room geometry. The result is a confusing and a worthless visualization. Like in the example, stream tubes, streamlines, moving particles, cross sections and isosurfaces all together usually do not work out properly. In this case, the primary aim of getting better understanding of some phenomena, can turn against itself. When a visualization is presented as stereoscopic images on a large back projected display, the result is more blurred than on a monitor. That is why the smallest and the thinnest details are often almost undescriptive. In a virtual room, it is better to keep the visualization as practical as possible.

Another issue is the requirement of real-time rendering. The elements, which are used in the visualizations, should not put too much load on the graphics system. Usually that leads to a compromise between the accuracy of the geometry and the frame rate. With very large models, one limitation is the amount of the memory but that has not been a problem in the BS-CAVE project.

In the next few subsections the chosen visualization elements are described more closely. Choices are made based on the testing of different options in the virtual room. The author has made most of the testing but other project personnel has been also heard, including the representatives from the participating companies.

### Paths of weightless particles

Perhaps the most suitable technique for visualizing a path of a weightless particle in a 3D flow field, is simply to show the trace of the particle with a spatial line or with an other line-like object. The path of a particle is also a good approximation for a real molecule in the air. In fact, the phenomenon called flow is a movement of molecules caused by a difference in energy potentials. Hence a the particle trace illustrates well the movement of the air mass.

The simplest and the computationally lightest element is a plain segmented line, but as described in the previous section, it is not a natural 3D object because of the fixed line width. To give the observer better depth cues, for example tubes should be used instead of the lines. A tube is like a swollen line and it is measurable in all three dimensions. When a particle trace is presented as a tube, it results in a flexuous pipe in the space. The shape alone does not tell much about, for example, the velocity of the particle. In the BS-CAVE project the varying color of the tube segments was chosen to present the values of the scalars. To present the velocity, the element must show not only the absolute value of the velocity (color) but also the direction of the flow. This is done by using arrowheads, or cones, with the tube element. The cones are placed on the junctions of the straight tube segments. The tube will have direction and it also appears smoother to the observer.

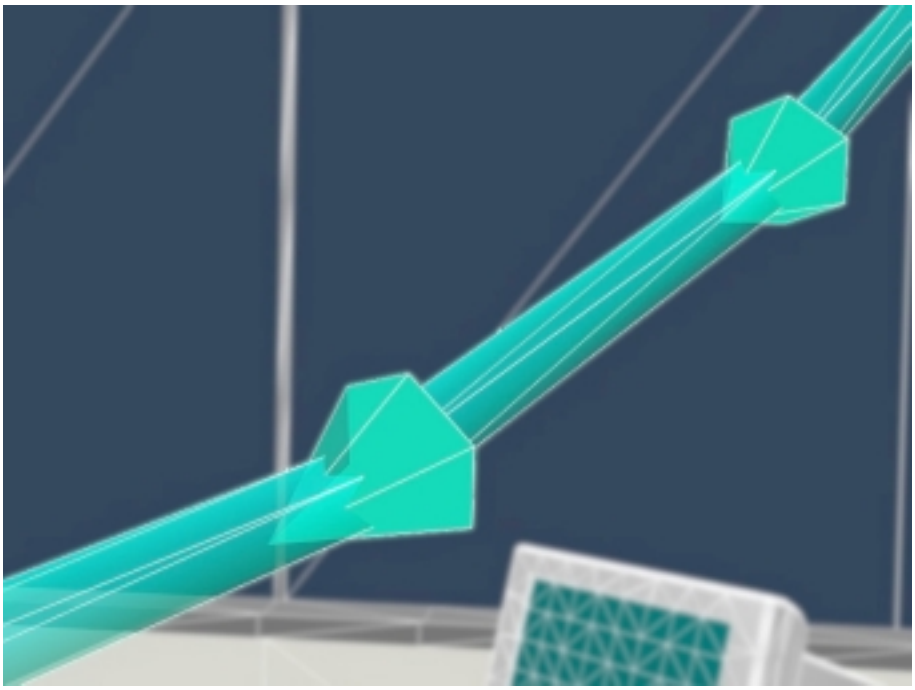


Figure 4.10: Polygonal structure of a tube and cones combination.

If the observer is interested in the path of only one particle, one tube manages the visualization well. However, usually the user wants to observe the behavior of a larger air stream, rather than one tiny particle. In this case one tube is not able to show how the airflow will spread, twist, or divide in many directions when it collides to an obstacle. To present the spreading of the stream, more than one element is needed. But the EVE is still a real-time rendering system and a tube with the cones is a relatively complex graphical object, at least compared to a plain line. A lot of polygons are needed for a rounded tube (see Figure 4.10). According to the real-time requirements, the number of the polygons in the scene should be kept as few as possible. This condition is against the idea of using multiple tubes in the scene. Further, if the field of vision is totally filled with the visualization elements, the rest of the scene will be blocked. A part of the tubes could be thinner, but the polygon count would remain the same. Again, there is a need for a compromise between the system performance and the quality of the visualization elements.

The solution in WolfViz is to generate multiple particle traces from several starting points, which are located very close to each other (see Figures 4.11 and 4.12). The middle trace is visualized with a colored tube with the cones indicating the direction of the flow. The outer traces are shown as simple colored lines. The coloring is based either on the velocity or the temperature scalar values, depending on the choice of the user. In addition, both the tube and the lines are slightly transparent to let the observer see also the photo-realistic geometry of the room behind the flow visualization. By using this technique, the geometry is kept relatively light but it is still able to illustrate the behavior of the real air mass. In Figure 4.13,

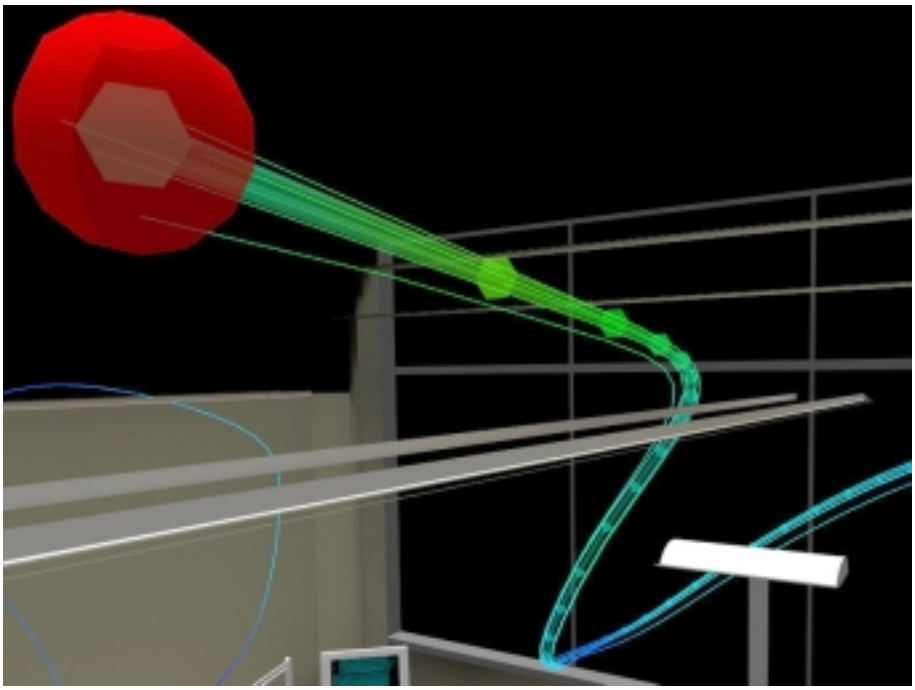


Figure 4.11: Starting region of particle traces inside the red ball.

the flow collides to a table stand and spreads apart. The room model will remain recognizable because of the transparency and the usage of a minimal amount of massive elements.

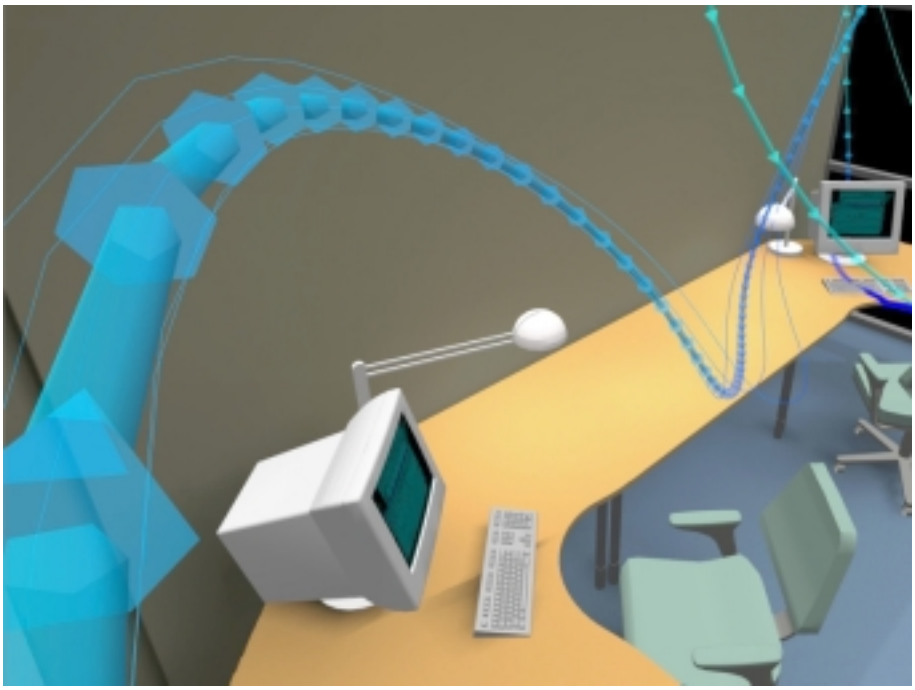


Figure 4.12: Tube and lines.

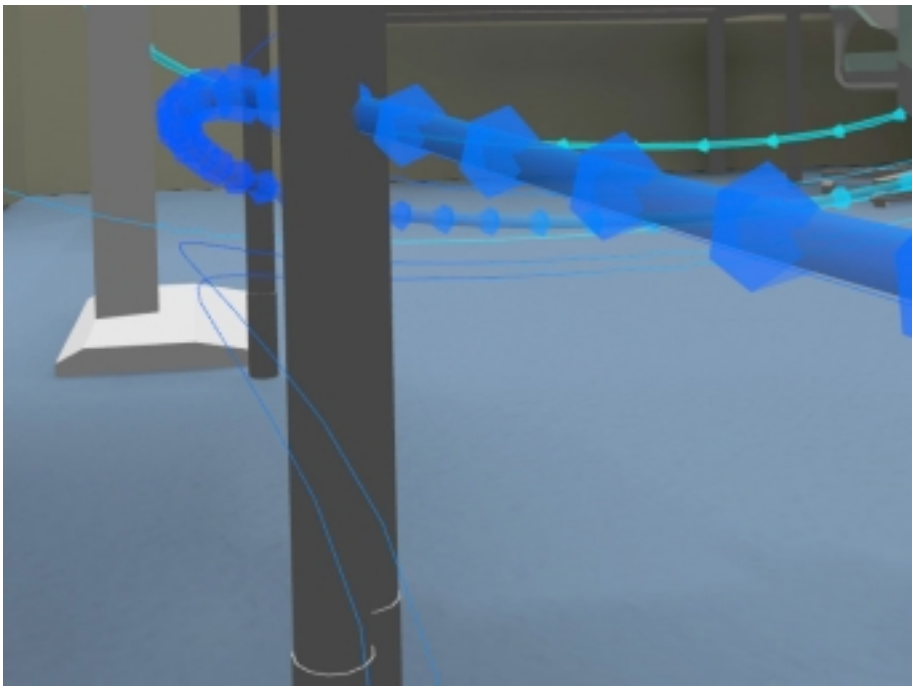


Figure 4.13: Flow spreading apart.

### Animated particles

Though, tubes and lines can be used to illustrate the velocity of the flow with a varying color, the relations of different velocities are better brought out by visualizing this particular attribute with animated particles moving along the paths in relatively correct speed. In a virtual environment, a rough comparison of the speeds of two moving particles, is often easier and more understandable than comparing two different shades of the same base color. Besides the convenience, it is always stimulating to get something lively in the virtual world and it makes the virtual experience more attractive and thus more immersive. Animated particles were also chosen to the collection of the implemented visualization elements.

When the user wants to observe extremely many particle traces at the same time, the alternative to use only moving particles instead of all the tube objects, generates less load to the graphics system and the usability of the visualization remains better. Again it is also possible that too many tubes and lines in the same scene may cause too much occlusion and the illustration suffers from the chaotic visualization (see Figure 4.14). In that case it will be better to use only the animated particles.

### Cross sections

Besides the velocities and the directions in different regions of the flow, the observer may be interested also in the scalar quantities. Scalars can not be visualized with vectors or animated

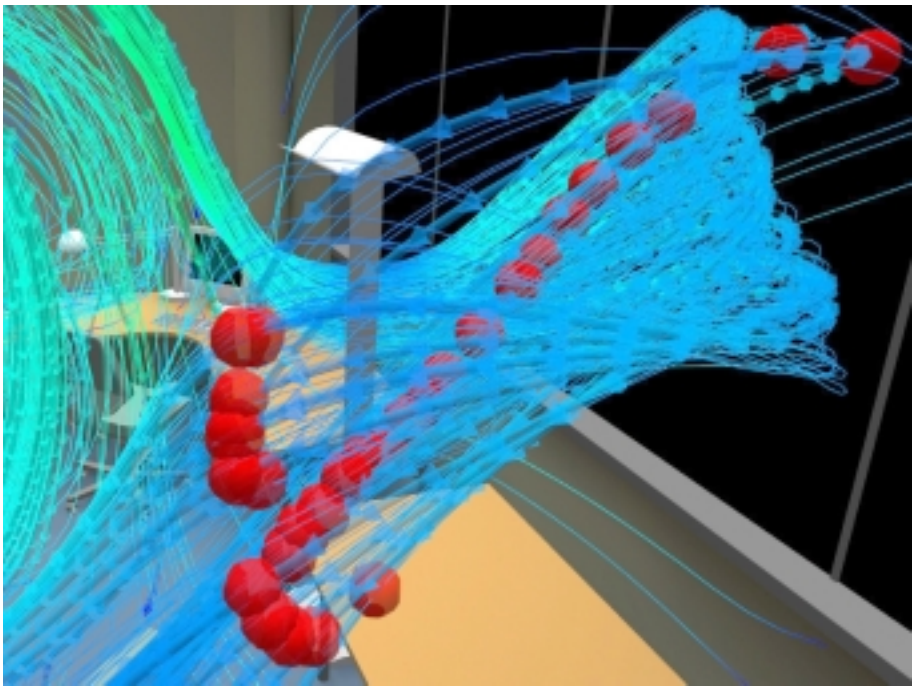


Figure 4.14: Thirty moving particles and their traces.

particles but other methods must be utilized. The tubes and the lines described in the previous sections may be colored in a way that certain color corresponds to a particular scalar value in that point. The idea of coloring an object to show the value of an invisible quantity, can be

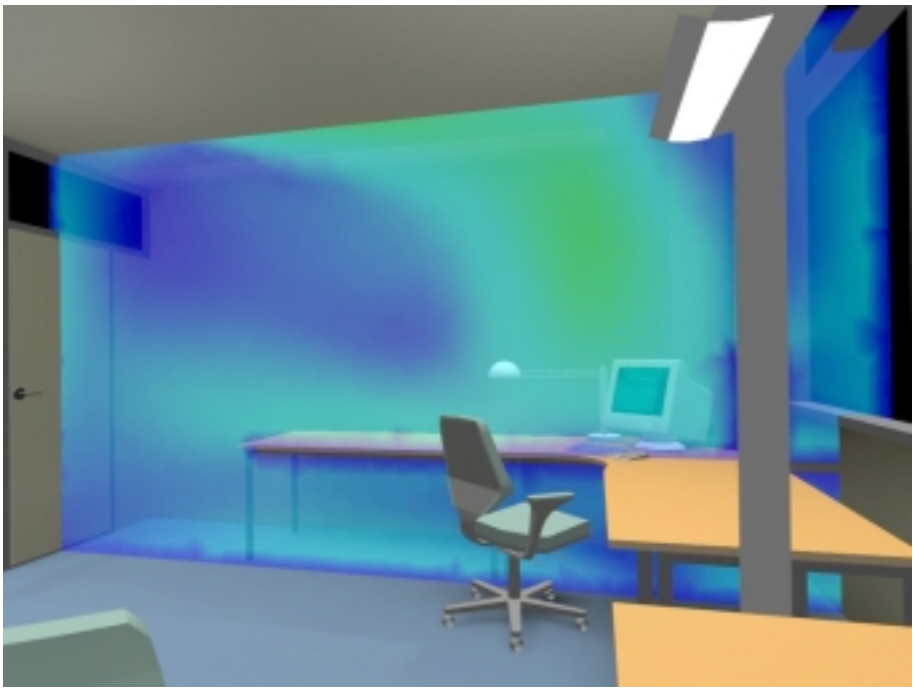


Figure 4.15: One colored cross section, color mapped to velocity quantity.

extended from the tubes and the lines to other geometric objects. The simplest object type for this purpose after a line is a plane. A colored plane can be used to illustrate the values on a cross section of a volume of information. In Figure 4.15 a plane is placed in a 3D model and it has been colored according to the absolute value of the air speed. Each point on the plane is colored according to the magnitude of speed on that particular location.

In WolfViz, the possible scalar values are the absolute values of the flow velocity and the temperature. There are a lot of other scalar values, which are often included in the results of CFD calculations (see Section 2.3.2), but they are not supported in the first version of WolfViz.

By using the colored planes or the cross sections, it is possible to get quickly a big picture about the conditions in the scene. The flow tubes show the path and the velocity of a particular particle trace but it may be difficult to adjust the starting point so that the path passes through a few specific points of interest. A plane surface can be set accurately to those points. In case of temperature, the observer is usually more interested in regions than only a few points. A plane surfaces serves better this need. The implementation of the WolfViz allows the observer to slide the plane surfaces across the scene. Thus it is easy to get a general view of the conditions caused by the temperature or the flow velocity.

#### Isosurfaces

The cross sections presented in the previous section suit well to overall sights of arbitrary regions, but if the observer is interested in any particular magnitude of a scalar value, there is still a need for another visualization technique. An isosurface illustrates the surface in the space where a scalar quantity has the same magnitude. The shapes of the isosurfaces are generated based on the results of the CFD calculations. The shapes may vary from tiny bubbles to an all scene filling mass, depending on the initial parameter.

In WolfViz, the scalars values, which can be used to generate isosurfaces, are temperature and absolute value of velocity. In the BS-CAVE project, the focus is on visualization of air flow and temperature, induced by a simulated ventilation system. If the software is used in the design of an air conditioning system, it is important that the designer can easily set isosurfaces to show the boundary values of a special interest and thus illustrate for example the pleasantness range of a room. In Figure 4.16 the isosurface corresponds to the flow velocity value of 0.25 m/s. In the region inside the surface the flow moves faster than the specified value and vice versa.

As the Figure 4.16 shows, the surfaces may be very large. In a scene a big isosurface blocks a lot of geometry behind itself. Therefore, isosurfaces should be rendered slightly transparent to let the scene be seen even there would be an isosurface between the observer and the

Figure 4.16: One isosurface for the flow velocity value of 0.25 m/s.

other objects. From the real-time graphics point of view, this may cause problems. The transparency and the great number of polygons have easily a degrading effect on the frame rate. Hence, some consideration should be used related to the illustration of the isosurfaces.

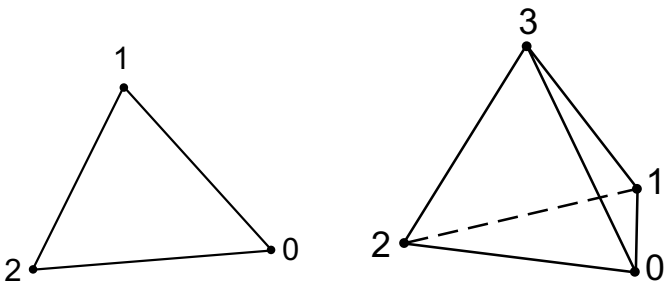
### 4.5.2 EnsightToVTK - CFD-data conversion

The flow data is produced as a result of the CFD calculations carried out by the BS-CAVE project participant, engineering office Olof Granlund Oy. The pre-processing and the execution of the actual solver part are done at Granlund's in a Windows NT environment and. The generated flow data is delivered in an EnSight Case file format to the EVE's SGI environment. The CFD software is CFX-5[22] produced by AEA Technology plc. EnSight[23] is a stand alone visualization software from CEI, Inc and EnSight Case is a file format designed for that software.

The EnSight Case file format for a CFD data is simple. CFX-5 produces eight ASCII files. One describes the content of the actual data files. One file contains all of the geometries of the model including the unstructured spatial tetrahedron grid and the surfaces of the objects. The surface geometry is described in triangles. Both primitives are referred also as cells. The rest of the files define the scalar and the vector values for all nodal points in the grid. The geometric primitives are shown in Figure 4.17 (a) and (b). The numbers in the figure present

[22] <URL:http://www.software.aeat.com/cfx/products/CFX-5.html> 25.5.2001
[23] <URL:http://www.ensight.com/> 25.5.2001

the vertexes whose coordinates are defined in the geometry file.




(a) Triangle. (b) Tetrahedron.

Figure 4.17: Geometric primitives that are used in the EnSight and the VTK unstructured grid formats.

### EnSight format

The idea of the EnSight format is first to define an array of points in the space by using the cartesian coordinate system and three coordinate values. All points are also given index numbers. After the definition of the points, the grid and the object surfaces are defined by referring to the indexed points. Unstructured tetrahedron grid is defined cell by cell by giving a cell an index number and four corner vertexes from the indexed point array. The same applies also to the triangle meshes but now the number of the vertexes is three.

Each nodal point has interrelated data of one or more scalar or vector quantities. Each partial data set is saved in its own file. The file contains the name of the quantity and a scalar or a vector array of values, one value for each earlier defined point. The EnSight file set related to the BS-CAVE project contains vector values presenting the direction and the velocity of the air flow. The scalar values are calculated for temperature, density, pressure, turbulence eddy dissipation, and turbulence kinetic energy.

### VTK unstructured grid format

The VTK ASCII unstructured grid data format uses mainly the same elements than the EnSight format. In the VTK's format all the information is in the same file. The file begins with a header data about the type of the file. The actual data consists of the definitions of the points, the cells, the cell types, and the scalar and the vector data. The point array defines the coordinates, which are later referred by the definitions of the cells. The cell types are in this case tetrahedrons and triangles, the same as in the EnSight format. The types are defined in an array, separately for every cell. After the definition of the geometry, the data arrays of

different quantities define the data values for every nodal point. In the beginning of every data array block, there is a definition for the type of the data. The number of different data arrays is not limited.

#### Conversion

To be able to manipulate and process the flow data in a VTK pipeline, the EnSight data must be converted to a form that can be read into a VTK application. VTK has support for multiple general types of data. The VTK unstructured grid data format is basically the same as the EnSight format. The conversion is straightforward and it is accomplished almost by only changing the order of the different data elements and by writing all data in one file.

The conversion software, EnsightToVTK, is written in the ANSI C programming language. When the conversion is run, all coordinate values of the points from the EnSight file are read in memory and the number of the cells in the file is calculated. The VTK file header containing the information about cell count is written out. All coordinate values for the nodal points are written after the header. After that, all elements defined by the point indexes, both the tetrahedron and the triangle cells, are read in and written out straight away and the types of the cells are defined in an array of identifiers. Finally, all data is read from different files and written out after headers lines.

The conversion is fast and simple. Nearly all of the time is spent reading the data in and writing it out. Calculations induces a minimal load.

### 4.5.3 Previz - From numbers to visual objects

The WolfViz software package is divided in three executables. The first is the EnsightToVTK conversion described in the previous section. EnsightToVTK prepares the flow data in a format that can be processed further with next software, Previz.

The division of the actual visualization process into two separate programs derives from a usability point of view. Even if the flow data was already produced and stored in the right file format, it takes relatively long time to generate the desired visualization elements. For example, from the data set, which was used in the project, the generation of one stream line takes approximately one second and the generation of a cross section takes about six seconds. Let it be assumed that the user wants to “slide” a cross section from one wall to the opposite to observe the changes in the behavior of the flow. Let the spacing between the cross section movement be ten centimeters. In a five meter wide room it would take five times ten times six seconds, in other words five minutes to do the operation. That is not acceptable for a task, which the user should be able to do in a few seconds. A solution for this heavy consumption

of time is to generate all visualization elements beforehand to the memory and just swap them fast in the scene when they are needed.

This solves the problems with the delays caused by the element generation while running the VR application. The solution is only partial, because it moves the delays in an earlier phase of the execution of the program. If the element generation is done directly after the program is started, it may take from several minutes to dozens of minutes before the actual interactive VR environment is ready to be used. That is not acceptable either. One way to minimize the inconvenience is to divide the visualization process in two separate software. The first program generates all the visualization elements, which the user is interested in, and saves the resulting geometric objects in a file. When the desired visualization is in the file, the second program just reads the file using only seconds of time and the VR environment is usable almost immediately. Because the visualizations are saved on the disk, the user may generate a lot of different visualizations to be used in different purposes. This solution is used in the BS-CAVE project.

From the implementation point of view, Previz uses three main components to produce the file containing the visualization of the input flow data. VTK is used to read and manipulate the data and to produce the geometric presentation of the visualization in VTK's own internal format. After that work is done, vtkActorToPF does the conversion from the VTK objects to the IRIS Performer objects. The final geometry is placed in an initialized IRIS Performer scene graph, which is then stored in a file.

### Command line options

Previz is currently implemented on the IRIX operating system and is run from a command line. When the program is run, the desired visualization is defined with command line options. The user can define 26 different parameters for the program. Five of them are the filenames and the paths of different input and output model files. Another five of them define the coloring, the model transformations, and the particle animation speed factor. The rest 16 parameters define the appearance of the actual visualization. The choices have effect on the different visualization techniques, the particle traces with the stream tubes and lines, the cross sections and the isosurfaces. In addition, the user may extract the object geometry from the flow model and present it in the final scene.

The starting points of the particle traces are defined by giving the end coordinates of a line from where the defined number of traces are started. For every trace, the particle path coordinates are stored on a file and a stream tube geometry is generated with the additional lines. For cross sections, the user defines how many sections are computed per a dimension. With this information each dimension is divided so that the cross sections are generated to constant distances from each other. It is also possible to generate the standard presentation

of the horizontal cross sections at heights of 0.1, 1.1, and 1.7 meters from the floor. For the isosurfaces, a range of the values and the number of the surfaces between these values are defined. Flow model objects are extracted from the data based on a cell range, which is also defined on the command line. This functionality requires specific knowledge of the input data format and the structure of the flow data model.

The coloring scalar identifier is defined for every visualization element. Coloring can be done independently for every element type, in a way that for example cross sections may present the values of the temperature while the color of the stream tubes are defined by the velocity values. A few command line options have effect on the storage of the visualization in Open Inventor file format. This feature is required to be able to transfer the visualization, or part of it, back to MS Windows environment. Because Postviz is not ported on that platform, it is not possible to interact with the visualization model in the same manners as it is done on the IRIX platform. That is why the user should define which part of the whole set of the visualization elements are stored in an Open Inventor (IV) file. The IV files can be converted to the VRML format and the flow models can be seen with any VRML compatible PC software. All included elements in the IV file are visible at the same time because switch nodes can not be used. That is why the elements, which will be stored in the IV file, have to be selected beforehand.

Why to define what to visualize? Why not just generate all the possible elements? It is important to delimit the element generation because of two reasons. The first is the limited amount of the memory. Both while the element generation and especially while running the VR environment, the application should fit completely in the memory. Otherwise the disk swapping will cut down the performance and the feel of the immersion decreases. Another issue is the time spent on the generation. It is not reasonable to keep the user waiting for at least minutes every time he wants to visualize only a minor part of all the possibilities e.g. only a few particle traces. By using the command line options, it is easy to specify which kind of visualization will the resulting file contain.

**Inside Previz**

When the Previz program is run, it first reads all the command line arguments and sets the internal parameters according to them. As described earlier, the parameters are used in many stages of the visualization element generation. After initializing IRIS Performer and a basic scene graph structure, the 3D model and the flow data is read in. The photo-realistic model is set in the scene graph with the required transformation to fit in with the flow data. If both the photo-realistic model and the flow data are produced using a same scale, system of coordinates, and positioning , the transformation is unnecessary. The temperature scalars and the velocity vector data are read and stored in the vtkUnstructuredGridReader-class objects.

One reader object contains the vector field and one scalar value set. In this case, the other scalar quantity is the temperature and the other is the absolute value of the velocity. In these objects the data is available for the rest of the VTK pipeline or pipelines. These objects provide also information about the data set, such as the value ranges.

The VTK part of the main element generation process can be conceptually separated in five different pipelines, which all begin from the reader objects. Even the pipelines are here presented individually, in practice they all begin from the same two optional reader objects. The connection depends on the chosen coloring quantity. The same pipelines are used in loops many times if the user has required multiple elements of the same type.

The first and the simplest pipeline is used to generate the outline object (see Figure 4.18). The figure illustrates a simple process where the data is first read in the memory by the reader object and then filtered somehow – which in this case refers to computing the outlines of the flow model. Here the numeric information about the bounds of the model is printed out for the user. Every pipeline, which generates geometric objects, ends with a suitable mapper object and an actor object.


Figure 4.18: VTK pipeline for outline geometry.

The program generates next the tubes and the lines that visualize the paths of the weightless particles. The pipeline is presented in Figure 4.19. It is more complex than the previous outline pipeline and it actually produces three different geometric elements, which are combined together. The process begins again with the reader, but another data source is also used. The vtkLineSource-object produces coordinates according to the given parameters for the starting points of the particles. These information sources are attached to the filter, which does the particle path integration. The resulting streamline is cleaned from overlapping coordinate points with the next filter. From this phase the pipeline branches in three sublines. In Figure 4.19, the middle line generates the tube around the streamline, the upper presents the pure line, and the lower adds the cones to the tube to illustrate the direction of the flow. Actually, the upper plain streamline visualization is used six times per one tube with a varying starting point to achieve the air mass visualization described in Section 4.5.1 on page 56. To present the particle traces as an animation in the final phase of the visualization, the paths of the particles are stored in files in a format supported by the IRIS Performer ’pfuPath’ functions. The ’pfuPath’ function collection is a part of the ’pfutil’ library and it provides a mechanism to move objects along a mathematically defined path. Each path is stored in its own ASCII text file to be read later into Postviz.

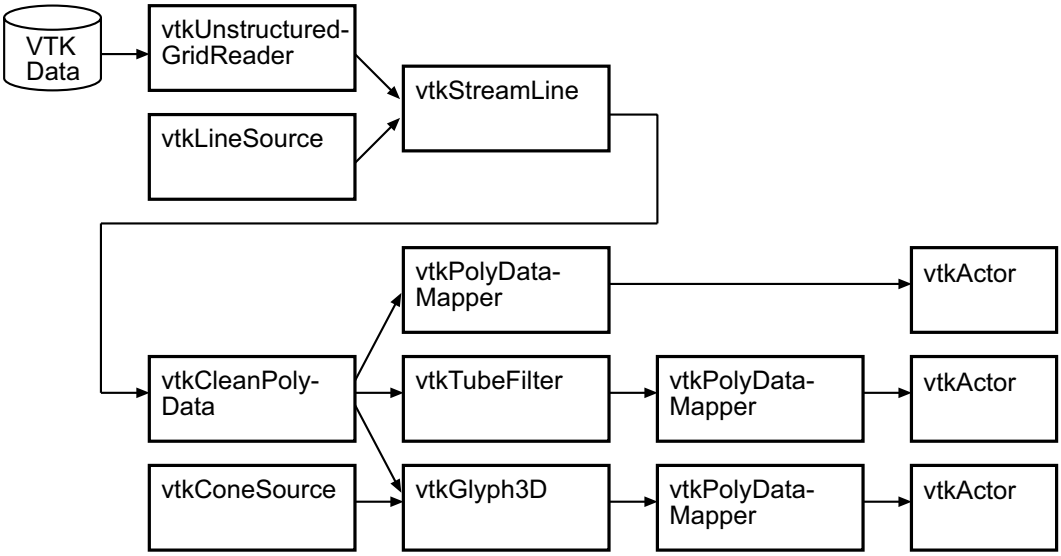



Figure 4.19: VTK pipeline for generation of stream tubes, stream lines and cones.

The cross sections are generated in two phases, which use the same pipeline (see Figure 4.20). The cross sections in all x, y and z directions are computed first according to the user defined spacing. They are followed by the standard horizontal cross sections. The VTK pipeline is almost as plain as in case of outline element. The cutter filter is only linked not just with the flow data, but also with the cutter function source, which is in this case a simple plane.

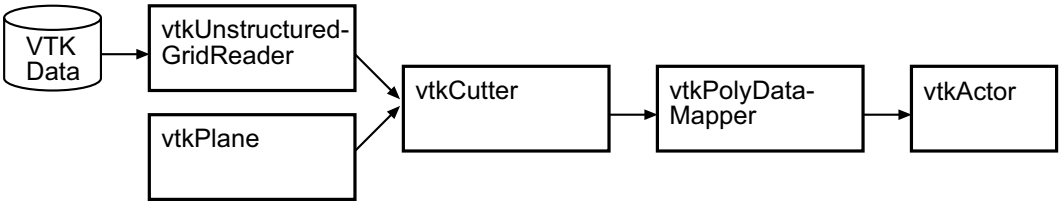



Figure 4.20: VTK pipeline for generation of cross sections.

The pipeline, which generates the isosurfaces, is again simple and straightforward (see Figure 4.21). The isosurfaces are generated by using a contour filter and the only additional feature besides the ordinary objects, is a filter that generates the vertex normals needed in Postviz for the proper rendering of the lighting.

The last visualization elements that are extracted from the flow data set in their own pipeline are the geometric objects in the flow model. The objects, such as walls and furniture, define the spatial attributes of the flow model. Besides that they fill some space in the model, they may have also an active role. For example, the computer monitors heat the air around them. Sometimes it is useful to see the temperatures on the surface of an object. The creation of the visualization of the temperatures on the surfaces of the recent monitor, is analogous to a

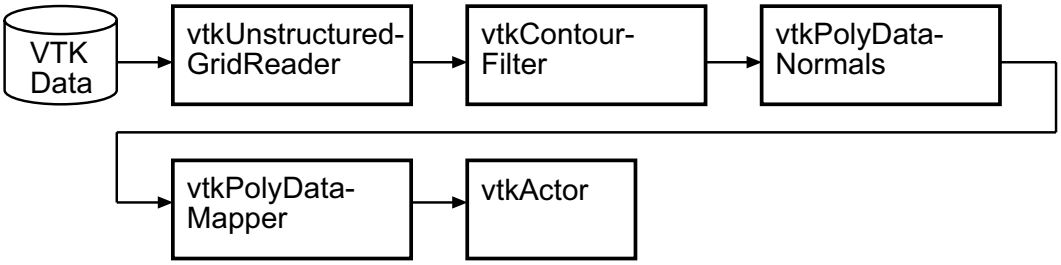



Figure 4.21: VTK pipeline for generation of isosurfaces.

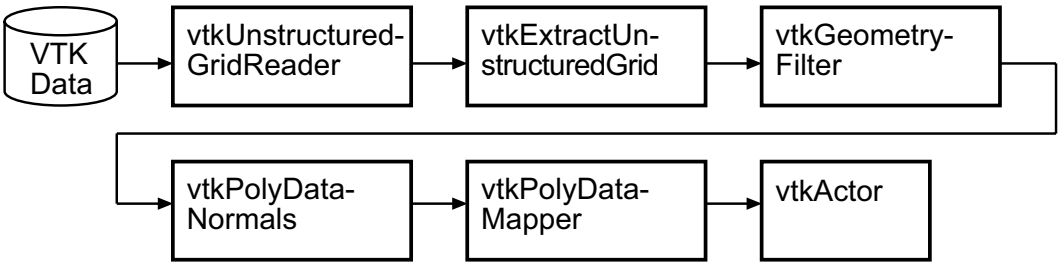



Figure 4.22: VTK pipeline for extraction of flow model geometry.

situation where many cross sections are combined and limited in such a way that they cover all the surfaces of that monitor. In other words, the flow model object visualization is not actually a new element type, but an application of the other methods and elements. Figure 4.22 presents the pipeline where certain cells that define the flow model object geometry are extracted from the data and converted in geometric VTK objects. Also here, the normals are computed to be used in the rendering phase.

The visualization objects are colored according to either the temperature or the velocity values of a certain range. For the users convenience, a scalar bar is provided to illustrate the correlation between the colors and the values (see Figure 4.23). Because the user can set the coloring ranges arbitrarily, the scalar bar must be generated dynamically for each visualization. Like all the other visualization elements, also the scalar bar is added to the scene as a 3D object. The generation process is separated in two phases. First the texture, which presents the color correlation, is generated. After that, the texture is wrapped around an oblong box to create a 3D scalar bar object. Both phases are done with the VTK objects. Texture generation is presented in Figure 4.24. The scalar bar texture have to be actually rendered on the screen with VTK's own renderer before it can be stored on the disk as a bitmap image. That is why the renderer objects are included in the pipeline. After the image is rendered, it is stored in a file by using a bitmap file format (BMP) exporter object. The pipeline is used twice in order to generate the textures both for the velocity and for the temperature quantities.

In the next phase, the bitmaps are used to texture 3D bars. Figure 4.25 presents the pipeline

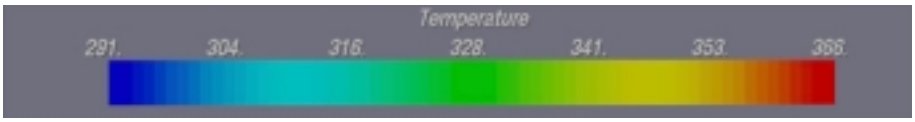



Figure 4.23: Scalar bar for temperature values (K).

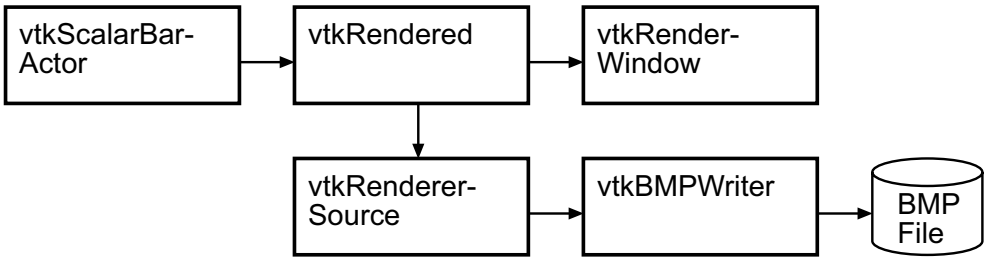



Figure 4.24: Generation of scalarbar texture.

from the BMP file and a cube source to the VTK actor. The texture is read from the file and mapped directly to the transformed cube geometry in the actor object.

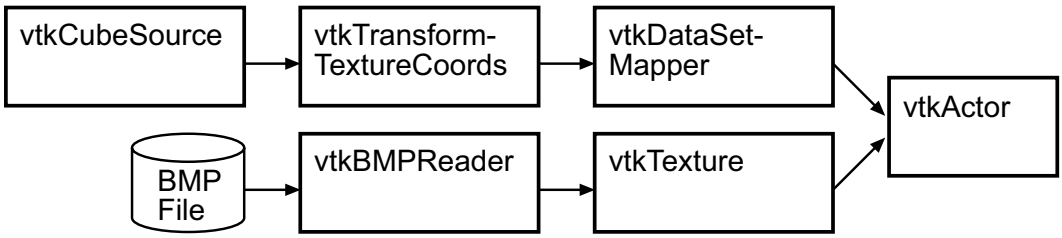



Figure 4.25: Texturing of a 3D box in order to create a scalar bar object.

After the visualization elements are generated and encapsulated in a corresponding vtkActor object, the actor objects are immediately converted to the IRIS Performer geodes by using the vtkActorToPF library function. The function does the conversion and returns a pointer to the geode. Thus the element may be added directly to the initialized IRIS Performer scene graph under a specific switch node. If the generated element is also specified to be stored in the IV file, it is added to the VTK renderer to be exported at the end of the program execution. The exporter part of the VTK pipeline is presented in Figure 4.26.

The IRIS Performer scene graph structure is illustrated in Figure 4.27. The light source lights the scene. The realistic model and the flow visualizations are attached directly to the root node. The realistic model part of the scene loaded from an OpenFlight file, begins from the 'pfGroup' node under the node named Real model. The small triangles under the visualization element nodes stand for a continuing scene graph. The detailed illustrations of those parts of the graph are presented in separate figures.

The following figures illustrate the structure of the scene graph below the nodes that are

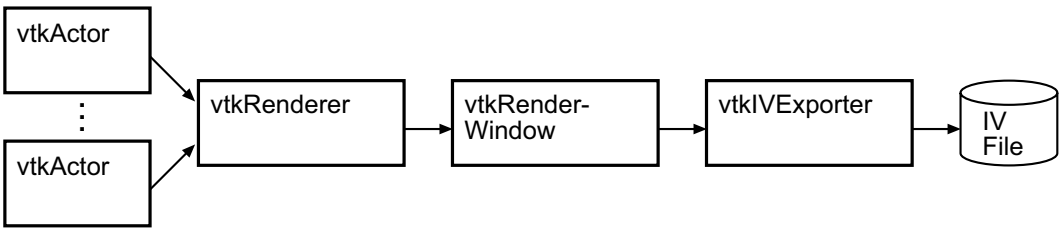



Figure 4.26: Exporting VTK actors to an IV file.

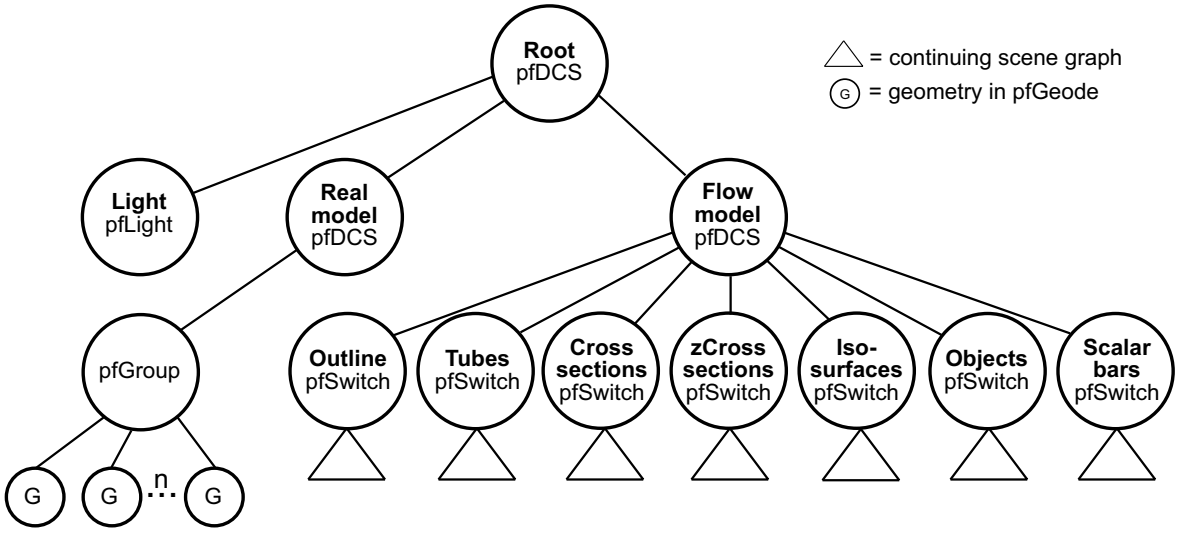



Figure 4.27: Top levels of the scene graph produced in Previz.

attached to the flow model head node. The 'pfGeode' nodes in the pictures are converted from the corresponding VTK actors and attached to the right parent node. The subgraphs of the outline and the flow model geometry illustrated in Figure 4.28 (a) and (b), are very simple. They consist of only the switch node and one 'pfGeode', which contains all the geometry. The graph for the isosurfaces is a bit more complex with its several 'pfGeodes' (see Figure 4.28 (c)). Each 'pfGeode' contains the geometry for one surface. The number of 'pfGeodes' depends on how many surfaces are generated for a particular visualization.

Figure 4.29 presents the subgraph, which illustrates the node structure of the stream tubes and lines. Each collection of a tube, cones, six lines and a ball is placed under a separate switch node. The ball illustrates the weightless particle. The collection of these elements is used to visualize one particle trace. The use of the switch nodes enables a flexible visibility control described further in the next section.

The double switch structure is used also with the cross sections to allow fast changes in the visibility (see Figure 4.30). All the cross sections made perpendicular to the same coordinate axis are placed in order under the same switch node in a way that each 'pfGeode' contains one cross section. In Figure 4.30, the cross sections are referred as planes. The sections in all

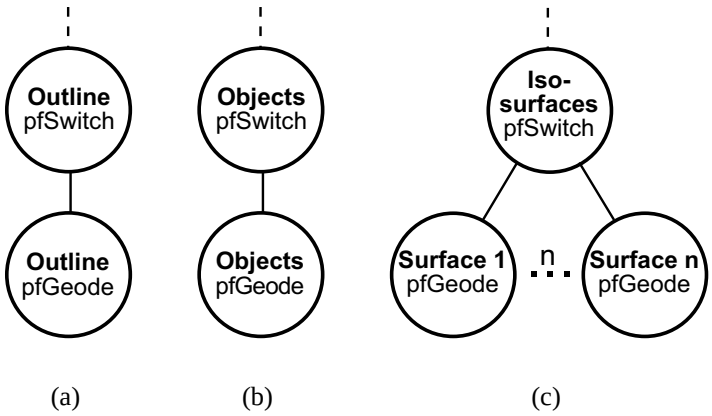



Figure 4.28: Detailed scene graph for outline (a), flow model objects (b) the isosurfaces (c).

three dimensions can be controlled independently. The standardized Z level cross sections are placed separately under their own 'pfSwitch' node. This structure is presented in Figure 4.31.

A billboard technique is used to make sure that the scalar bars are viewed always from the right direction. A billboard is an element, which automatically turns towards the camera in the rendering phase. In IRIS Performer, a certain billboard node manages the orientation dynamically. Figure 4.32 illustrates how the billboard controlled geometry is placed in the scene graph. Actually, this location is valid only before the interactive visualization in Postviz. In the initialization phase of Postviz, the scalar bars have to be moved under the

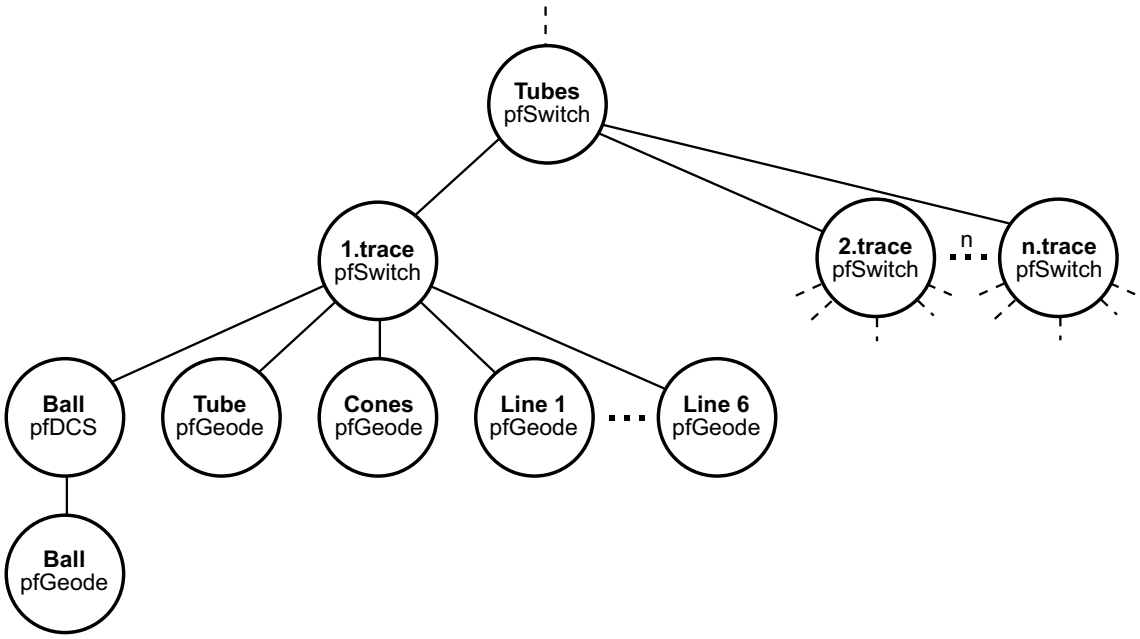



Figure 4.29: Detailed scene graph for stream tubes and lines.

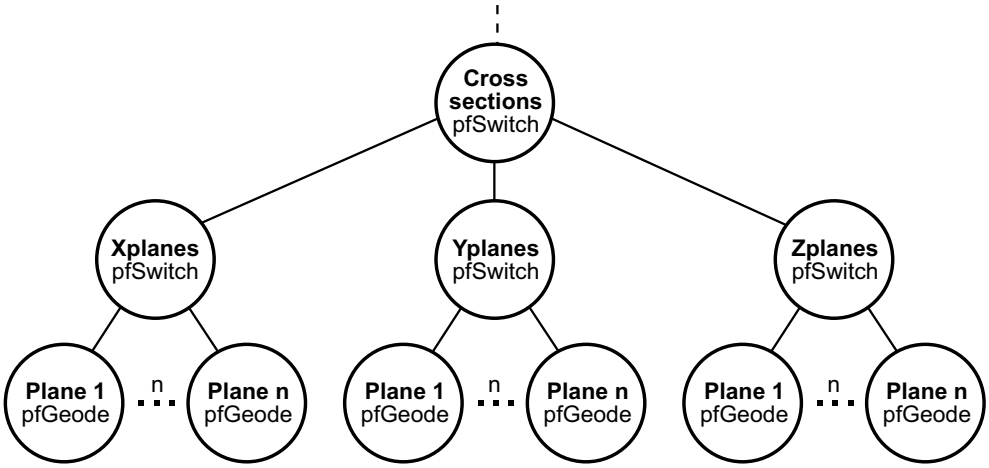



Figure 4.30: Detailed scene graph for cross sections.

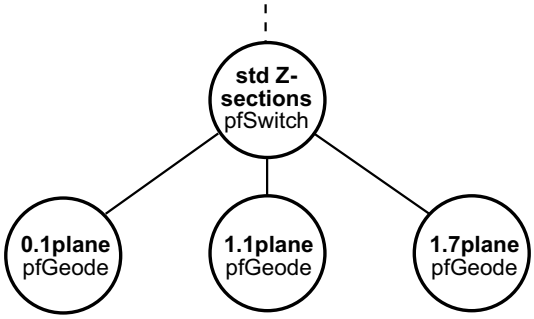



Figure 4.31: Detailed scene graph for standard Z cross sections.

whole scene root to enable proper translation control.

### 4.5.4 Postviz - Interaction

When the flow data is processed so far that all the visualization elements are generated and stored in a file, the final visualization can be realized. Postviz is the third software in the WolfViz family. It reads the geometric data from a file and provides an ability to interact with the scene. Postviz is based on the VR Juggler virtual reality framework (see Section 4.3.4) and it uses IRIS Performer (see Section 4.3.2) as the rendering engine. VTK is not needed anymore, in Previz it was used to generate the geometric objects, but after the elements were converted in IRIS Performer format, the work of VTK was done.

#### Command line options and initialization

Postviz is built on IRIX operating system and it is started from an UNIX shell command line. Postviz provides six command line options, although most of the visualization controls

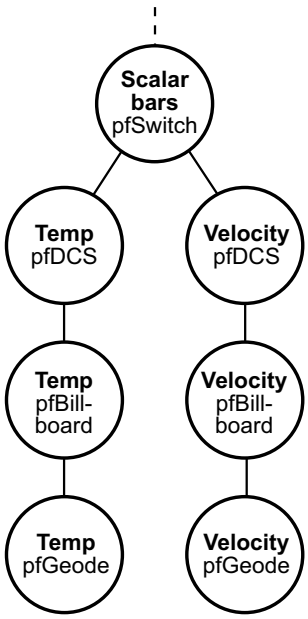



Figure 4.32: Detailed scene graph for billboard controlled scalar bars.

are managed interactively while the program is running. Three of the command line options define the model file, which is to be loaded, and the directories where the model textures and the particle trace path files are to be found. Two options allow the user to redefine the model transformation and the speed coefficient for the weightless particles animation. The last option has effect on the IRIS Performer texture rendering. By default, texture rendering is optimized for maximum performance. In very detailed and gradually colored textures the optimization may be seen as undesirable artifacts and overall weakened texture quality. It is possible to redefine IRIS Performer internal texture format so that the rendering quality corresponds to the actual texture quality. In other words, it increases the bit depth from 16 to original 32. The drawback with the redefinition is sometimes noticeable loss in performance. However, Postviz provides this feature to be used when it is possible.

When the program is started from the command line, it needs also configuration files as command line arguments to set up the VR Juggler environment. The environment definition consists of for example displays and physical or simulated input devices. By changing the files that define the environment the same application can be run in different machines with a different physical configuration.

In the beginning of the program execution, a VR Juggler kernel object is created and the environment is set up according to the chosen configuration. Postviz uses keyboard as the input device, which is initialized in this phase. The other command line options are also read, and the corresponding internal application variables are set accordingly. The kernel is started and the application is set to be controlled by the kernel. IRIS Performer is initialized, the visualization model is loaded from the file, and it is attached under the root node of the

scene graph. The important switch nodes are parsed from the scene graph so that they can be referred when needed. After this, if required, the internal texture format is redefined as described earlier. The particle paths are loaded from the files to be used later in the animation. The scalar bars subpart of the scene graph is moved directly under the root node. Without the move operation, the scalar bar control would need additional computation for compensating the model transformations caused by the navigation maneuvers. The idea is to keep the scalar bar in front of the user without dependencies to the model location or orientation. After the parameters and the numerical features of the flow model are printed out to the user, the short initialization phase is completed and the actual visualization application is started.

**Interaction with the scene**

The interaction between the user and different visualization elements is carried out by using an ordinary keyboard in a way that a certain keypress induces a certain type of visualization element to change. Each element type has its own control keys, usually one for forward selection and another for backward selection. It is not the right method from the immersion point of view, but designing and implementing a totally new and proper interaction technique would be a demanding project itself. In Postviz, each visualization types, such as the stream tubes and the cross sections, are individually controllable and the different combinations can be freely chosen.

One particle trace is visualized by using a stream tube, six lines and an additional sphere object, which presents the notional particle. The whole model may contain many particle traces. From these traces, the user may choose none, one or all of them to be seen at a time. The user may also choose to see the whole stream visualization or only the particles. The latter option is usable especially with the particle animation. It is also practical to change the start point forward and backward along the line of the starting points in a way that the visible stream tubes change gradually. If the start point spacing was in the generation phase dense enough, the illusion of moving or sliding the starting point is very convincing.

The animation is also controllable. The particle animation can be started, stopped, continued and the particles can be reset back to the starting positions. After the reset the animation can be started again from the beginning.

Cross sections may also be selected, or slid, from side to side or from up to down and vice versa, depending on the orientation of the sections. Each direction, X, Y and Z, can be handled individually to present none, one or all the cross sections of that orientation. The standard Z levels cross sections can also be chosen almost on the same principle. The difference is that the sliding can be done only upwards.

The isosurfaces are selected with the same principles as the particle traces but, now the

forward and backward selection changes the surface to correspond with a bigger or a smaller value of certain quantity.

The scalar bar can be used to inform the user about the correspondences between the colors of the visualization elements and the actual quantity values. The scalar bar is defined in a way that it will appear as a 3D object always in front of the user and when the user position is changed it moves relatively. Thus it is independent from the rest of the scene. The interaction with the scalar bar is managed with one key. The first key press brings the scalar bar in the scene with a color range of the velocity values. Second press changes the quantity and values to illustrate the range of the temperatures in the model. The third press switches the visibility off to wait for the next appearance.

The features above are useful in any normal operating situation. To be able to adjust the coordinate systems of the flow data model and the realistic model to be congruent, Postviz provides a possibility to view and hide the flow data outlines. With this feature, the user can see if the positions and orientations of the two models correlate with each other. The possibility to see the geometry of the flow model may also be used to verify the similarities of the models. In a general case, the flow model and the realistic model objects have coplanar polygons. It causes annoying flickering and thus these flow model objects usually can not be added to the scene. On the other hand, if the flow model geometry is defined to enclose entirely the realistic model objects, but still conserving the shapes accurately, the flow model objects are a convenient way to illustrate for example the temperatures on the object surfaces. Both outlines and objects are toggled between the visible and the invisible by using dedicated keys on the keyboard.

# Chapter 5

# Results and evaluation

## 5.1 Mixing the visible and the invisible

One partial objective in the BS-CAVE -project was to present the flow visualization together with a photo-realistic room model in a 3D virtual environment. Generally speaking, it can be said that the results met the requirements. The sample data used in the project, included a realistic room model and a flow data set, which was generated by a CFD software as a result of a simulation. The room model was successfully transferred to the EVE, the data set containing the flow field of the room was visualized using multiple graphical elements, and the combination of the room model and flow visualization turned out to be a functional solution. In fact, using the room model as an environment for flow visualization really helps in understanding the actual purpose of certain configuration of ventilation equipment.

Criteria for the implementation was set for the quality of the visualization, the usability of the system, and for the immersive experience in the EVE. The quality of the visualization itself is completely comparable to many other visualization systems. Of course, the visualization possibilities are not as versatile as in commercial products, but the available options do work well. Part of the elements introduced in Section 2.3.3 were not chosen, because of the implementation of those elements would have required more effort than the project had resources for (special glyphs and magic lenses), or they were evaluated to be unsuitable for 3D presentation in a VR environment (pure arrow plots). They also satisfied the interests of the project participants. The chosen objects, stream tubes with additional cones and lines, spheres in particle animation, and slightly transparent cross sections and isosurfaces all succeed to support the sense of space.

The usability of the WolfViz-software needs improvement in many areas. The pre-generation of visualization elements and their storage in the memory provides fast run-time access to different visualization variations. The requirement for the smooth interaction was thus

achieved, but the interface through a keyboard should be changed to a more natural VR interface such as a glove and a magnetic tracker. Also, in the preparation phase, in which the data is converted and the visualization elements are generated, a normal UNIX command line acts as an interface to the user. This solution is functional after short training, but is still a bit clumsy and needs improvement.

The demonstrations of the implementation were carried out using stereo imaging and three walls of the EVE. Thus the field of view was 270 degrees on a horizontal plane, which provided a very immersive environment. The tests with three walls left the testers with a feeling that floor projection would improve the immersion considerably. Anyway, all users have commented the environment being immersive even at the current stage. The comments from the project participants have been very positive. In the evaluation of the visualization itself, it should be noticed that unaccustomed users are often very impressed by the overall VR experience, and they may have difficulties on concentrating on the quality of the visualization itself.

## 5.2 Performance

Performance is always an issue. The users do not want to wait for the application to respond or to finish its tasks. The faster the application works, the happier the user. The WolfViz software family consists of three programs. The first is the flow data conversion software. It has to be run only once for every new data set and therefore its speed is not critical. In case of 32 MB EnSight data, the conversion takes about twenty seconds and produces 22.5 MB of data in the VTK format.

The second program, Previz, is more interesting from the performance point of view. The user may want to create many different visualizations from the same data set and the program must be run many times. With the SGI Onyx2 computer and the test data set related to the project, computing one stream tube with six lines takes about five seconds. One cross section takes about six seconds, and one isosurface about twenty seconds to generate. Time is also consumed on reading and writing the data in and out etc. For example generating twenty particle traces, thirty cross sections and five isosurfaces take about seven minutes in all. Response times of this magnitude really matter, when the user wants to run the visualization generation multiple times while trying to find the best solution.

The most critical performance issues are related to the last program, which renders the scene on multiple displays, creates the virtual reality experience, and provides interaction between the visualization and the user. Table 5.1 shows the frame rates for different visualization cases. All visualization elements are generated from the same sample data set. The room model itself consists of 28200 triangles and 15 MB of textures. Stream element refers to a

combination of a tube, six lines and cones illustrating the direction of the flow. The test was run on three walls on stereoscopic graphics mode. Each wall was run with a resolution of 1024 times 1024 pixels and the textures were rendered with higher quality than the standard IRIS Performer settings define. The performance test showed that the frame rate drops considerably as the complexity of the scene increases.

Table 5.1: Performance test results for sample data visualization rendered on three walls in stereo.

| Visualization | Frame rate |
|---|---|
| Office room model | 15 fps |
| Room + 30 particles | 15 fps |
| Room + one particle and a stream element | 12 fps |
| Room + one cross section | 8.5 fps |
| Room + three cross sections | 4 fps |
| Room + isosurface | 3 fps |
| Room + 30 particles with 30 stream elements | 1.3 fps |

The overall system performance reflects to the amount of rendered images in a second, or frames per second (fps). Thus, the frame rate is a compact form for presenting measurements of a system performance. For visual and dynamic 3D applications, an average requirement for the minimum frame rate is ten frames per second [Bryson 1994]. The achieved frame rates with the model room alone, the room and one stream element, and the room with thirty animated particles, are reasonable. The drop while rendering one cross section was on the limit of acceptance, but multiple cross sections and even one isosurface caused the performance to drop under convenience level. It would be an important feature to be able to use isosurfaces and still keep the frame rate high enough. One solution would be a more sophisticated optimization of isosurface objects as well as other visualization elements. The future releases of the VR Juggler software framework may also provide better performance.

## 5.3 Problems and challenges

As always, a few problems and challenges appeared during the project. Part of them were related to the technical issues while the rest were more general process-related issues.

One real problem in the beginning was that the project had to start with a search and evaluation of suitable software tools. While the evaluation was still ongoing, the implementation of the navigation part of the project had already started. That is why the overall planning for the integration of the visualization and navigation was neglected in the early stage of the project. Fortunately the software platform was chosen finally to be the same on both parts of the project, and the integration was eventually not a problem at all. The consolidation

challenges of compiling several software libraries in the same application were also solved after a hard work.

A minor defect was that there was only one 3D model with a corresponding flow data set available. In fact, the flow data was not completely consistent with the room model, but it was still possible to carry out the project. During the implementation, there were plans to produce another model with a data set, but new CFD calculations did not fit in the project schedule. The differences between the room model and the flow data geometry caused for example stream tubes penetrating through the furniture and the ceiling.

Part of the problems were related to the EVE facility itself. The extension project, which aimed to make the facility capable of using four projection surfaces, began in the middle of the BS-CAVE project. It caused several perioids of a few days when the system was not in use at all. Delays were also caused by the video projectors, an accident with the screen canvas, and synchronization problems related to display formats.

It could also be said that in one sense the project turned against itself at the point when the first software version was completed but writing of this thesis was not even near the end. Incalculable amount of demonstrations of the extended EVE and other minor tasks together took a lot of resources from finalizing this work.

# Chapter 6

# Conclusions

This work indicated that a combination of a realistic room model and an air flow visualization results in an viable solution. When the whole visualization is presented in a spatially immersive virtual environment, the sense of the space and the spatial relations in the model are experienced stronger. The BS-CAVE project encouraged us to keep the development work going and strengthened the confidence of the project participants in virtual reality (VR) technology.

Generally speaking, the visualizations of air flow simulations can be used efficiently as a part of a building design process. The additional value of a virtual environment as a presentation facility for the design expert himself, should be researched critically in the future. However, VR is a valuable tool when different information and designs are presented to experts of other professions, or to the customers. VR is also a convincing tool, which can be used as a help in decision-making. A practical example is visualizing the lighting conditions inside a building. The customer hardly understands the curves of distribution of the luminous intensity, but when a realistic 3D visualization is used, the design becomes completely clear. Also the convenience of air conditioning can be illustrated by spatially presenting the areas where the room temperature and flow velocity are acceptable.

In this thesis the fundaments of scientific visualization and virtual reality technology have been discussed. In the applied part of the thesis these two fields have been integrated in a software implementation called WolfViz. The selection of the software tools, which were used in the implementation, have been described briefly. The features of the chosen software systems and the functionality of the own implementation has been discussed in more detail. Finally, the results of the visualization of flow data in photo-realistic virtual environment have been presented and evaluated.

## 6.1 Status quo

At the current stage, the EVE facility and the WolfViz software – integrated with the navigation system – can be used for impressive immersive demonstrations. The usage of the visualization software requires a trained user but when the visualization has been prepared, any user can run it in the EVE and navigate in the model. The EVE facility is still lacking the floor projection but at the time it will be added, the immersion will be even stronger.

## 6.2 Future work

It is highly probable that the laboratory will start a follow-up project to develop the VR techniques further. The focus will still be on the building services. The main objectives in the follow-up project will be on integrating the audio features to the visualization system, developing new interaction techniques, and visualizing other typically non-visual building service information in a photo-realistic room. The new interaction refers for example to the possibility to easily compare different lighting conditions in a room.

The flow visualization software also needs improvement. More quantities, in addition to the velocity and temperature, will be visualized. If the computation times turn out to be too long, parallelization of the generation process will make the response times shorter. It would be more convenient, if the features currently managed through command line options, would be available through a graphical user interface. More flow data sets are needed to prove the versatility of the software. The visualization control methods in Postviz should be re-implemented in such a way that the user could interact with the scene from the EVE by using VR input devices, not the keyboard.

In the distant future when the computation power is increased by a few decades, it will be possible to generate the visualizations dynamically while the rendering software is running. At that time the features of Previz can be integrated in the real-time part of the current Postviz.

One challenge in the beginning of the BS-CAVE project was that the software tools had to be found, tested, and learned first. That is why it was hard to define precise specifications for the implementation. The tools and and the software environment are now well defined. Therefore the overall planning in the follow-up project will be easier. The planning should also cover things like coding style, interfaces between objects, and the use of design patterns.

The industry certainly is interested in virtual reality applications. However, in the future, one of the most important practical issues is to find such methods that the models and data sets can be transferred automatically from the designers' working environments to be visualized

in EVE. Only then can the high-end virtual environments be integrated seamlessly to the industrial process to provide additional value with moderate costs.

# Bibliography

[Allison et al. 2001]

Allison, R., Harris, L., Jenkin, M., Jasiobedzka, U., Zacher, J. (2001). *Tolerance of Temporal Delay in Virtual Environments*, Proceedings of IEEE Virtual Reality 2001, Yokohama, Japan, March 2001, pp. 247–254.

[Berhhers and Feiner 1994]

Beshers, C.G., Feiner, S.K. (1994). *Automated Design of Data Visualizations*, in Scientific Visualization, Advances and Challenges, Rosenblum, L., Earnshaw, R.A., Encarnação, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D., Editors, London, UK, 1994, pp. 87–102.

[Bierbaum 1998]

Bierbaum, A. and Just, C. (1998). *Software Tools for Virtual Reality Application Development*, SIGGRAPH 98 Course 14 Applied Virtual Reality Notes, ACM SIGGRAPH 1998, Orlando, Florida, USA, 1998, 44p. Also available on-line at `http://www.vrjuggler.org`

[Bierbaum 2000]

Bierbaum, A. (2000). *VRJuggler: A virtual platform for virtual reality application development*, Master's Thesis, Iowa State University, Ames, Iowa, USA, 2000.

[Bryson 1994]

Bryson, S. (1994). *Real-Time Exploratory Scientific Visualization and Virtual Reality*, in Scientific Visualization, Advances and Challenges, Rosenblum, L., Earnshaw, R.A., Encarnação, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D., Editors, London, UK, 1994, pp. 65–85.

[Card et al. 1999]

Card, S.K., Mackinlay, J.D., Shneiderman, B. (1999). *Readings in Information Visualization, Using Vision to Think*, Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999. pp. 1–34.

[Carpendale et al. 1997]

Carpendale, M.S.T., Cowperthwaite, D.J., Fracchia, F.D. (1997). *Extending Distortion Viewing from 2D to 3D*, IEEE Computer Graphics and Applications, July/August 1997, pp 42-51., also in Readings in Information Visualization, Using Vision to Think, Card, S.K., Mackinlay, J.D., Shneiderman, B., Editors, Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999. pp. 368–379.

[Chen 1999]

Chen, P. C. (1999). *Developing a Personal Computer-Based Data Visualization Using Public Domain Software*, Proceedings of SPIE 1999, Visual Data Exploration and Analysis VI, San Jose, California, USA.

[Cruz-Neira et al. 1993]

Cruz-Neira, C., Sandin, D.J., DeFanti, T.A. (1993). *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*, ACM Computer Graphics, Vol. 27, Number 2 , July 1993. pp. 135–142.

[de Leeuw and van Wijk 1993]

de Leeuw, W.C., van Wijk, J.J. (1993). *A probe for local flow field visualization*, in Proceedings Visualization '93, IEEE Computer Society Press, Los Alamitos, CA, 1993. pp. 39–45.

[Earnshaw and Jern 1994]

Earnshaw, R.A., Jern, M. (1994). *Fundamental approaches to interactive real-time visualization systems*, in Scientific Visualization, Advances and Challenges, Rosenblum, L., Earnshaw, R.A., Encarnação, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D.,Editors, London, UK, 1994, pp. 223–238.

[Eckel 1997a]

Eckel, G. (1997). $IRIS\ Performer^{TM}$ *Programmer's Guide*, SiliconGraphics Computer Systems Product Manual, SGI Document Number 007-1680-040. Silicon Graphics Inc., USA, 1997, 698p. Also available on-line at `http://www.sgi.com/software/performer/manuals.html`

[Eckel 1997b]

Eckel, G. (1997). $IRIS\ Performer^{TM}$ *Getting Started Guide*, SiliconGraphics Computer Systems Product Manual, SGI Document Number 007-3560-001. Silicon Graphics Inc., USA, 1997, 276p. Also available on-line at `http://www.sgi.com/software/performer/manuals.html`

[Feiner and Beshers 1990]

Feiner, S., Beshers, C. (1990). *Worlds within Worlds, Methaphors for Exploring*

*n-Dimensional Virtual Worlds*, originally published in Proceedings UIST'90, pp 76-83, also in Readings in Information Visualization, Using Vision to Think, Card, S.K., Mackinlay, J.D., Shneiderman, B., Editors, Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999. pp. 96–106.

[Foley and Ribarsky 1994]

Foley, J., Ribarsky, B. (1994). *Next-generation Data Visualization Tools*, in Scientific Visualization, Advances and Challenges, Rosenblum, L., Earnshaw, R.A., Encarnação, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D., Editors, London, UK, 1994, pp. 103–125.

[Fuhrmann and Gröller 1998]

Fuhrmann, A., Gröller, E., (1998). *Real-Time Techniques For 3D Flow Visualization*, IEEE Visualization'98 Proceedings, 1998, pp. 305–312.

[Gröhn et al. 2000]

Gröhn, M., Jalkanen, J., Laakso, M., Mantere, M., Forsman, S., Heljomaa, K., Salonen, T. (2000). *3D Visualization of lighting design in HUTCAVE virtual environment*, Nordisk Belysningkongress 2000, 13.-15.9.2000, Helsinki, Finland.

[Hearn and Baker 1997]

Hear, D., Baker, M.P. (1997). *Computer Graphics C Version, 2nd ed.*, Prentice-Hall, Inc., New Jersey, USA, 1997, 652 p.

[Hiipakka et al. 2001]

Hiipakka, J., Ilmonen, T., Lokki, T., Gröhn, M., Savioja, L. (2001). *Implementation issues of 3D audio in a virtual room*, Proceedings of 13th Symposium of IS&T/SPIE, Electronic Imaging 2001, Vol. 4297B, San Jose, California, USA, January 21-26, 2001.

[Hänninen 1999]

Hänninen, R. (1999). *LibR - An object-oriented software library for realtime virtual reality*, Licentiate Thesis. Laboratory of Telecommunications and Multimedia, Helsinki University of Technology, Espoo, Finland, 1999.

[Iwata et al. 2001]

Iwata, H., Hiroaki, Y., Nakaizumi, F. (2001). *Gait Master: A Versatile Locomotion Interface for Uneven Virtual Terrain*, Proceedings of IEEE Virtual Reality 2001, Yokohama, Japan, March 2001, pp. 131–137.

[Jalkanen 2000]

Jalkanen, J., (2000). *Building A Spatially Immersive Display: HUTCAVE*, Licentiate

Thesis, Laboratory of Telecommunications and Multimedia, Helsinki University of Technology, Espoo, Finland, 2000.

[Just 1998]

Just, C., Bierbaum, A., Baker, A. and Cruz-Neira, C. (1998). *VR-Juggler: A Framework for Virtual Reality Development*, Proceedings of 2nd International Immersive Projection Technology Workshop, Ames, Iowa, USA, 1998. Also available on-line at `http://www.vrjuggler.org`

[Kalawsky 1993]

Kalawsky, R. (1993). *The Science of Virtual Reality and Virtual Environments*, Addison-Wesley, Cambridge, UK, 1993. 405 p.

[Kaufman 1994]

Kaufman, A. (1994). *Trends in Volume Visualization and Volume Graphics*, in Scientific Visualization, Advances and Challenges, Rosenblum, L., Earnshaw, R.A., Encarnação, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D., Editors, London, UK, 1994, pp. 3–19.

[Keller and Keller 1993]

Keller, P.R., Keller, M.M. (1993). *Visual Cues, Practical Data Visualization*, Institute of Electrical and Electrics Engineers, Inc., 1993. 229 p.

[Koponen 2001]

Koponen, J. (2001). *FEMin viisi askelta*, Tietoyhteys 1/2001, CSC, Espoo, Finland, 2001, pp. 22–25.

[Kramer et al. 1999]

Kramer, G., Walker, B., Bonebright, T., Cook, P., Flowers, J., Miner, N., Neuhoff, J., Bargar, R., Barrass, S., Berger, J., Evreinov, G., Gröhn, M., Handel, S., Kaper, H., Levkowitz, H., Lodha, S., Shinn-Cunningham, B., Simoni, M., Tecumseh Fitch, W. and Tipei, S. (1999). *Sonification Report: Status of the Field and Research Agenda*, ICAD, Report was prepared by an invited interdisciplinary group of researches gathered at the request of the National Science Foundation, 1999.

[Laakso 1999]

Laakso, M. (1999). *More Software Tools for Virtual Reality Application Development (Maverik, LibR and Open Inventor)*, Internal report, Laboratory of Telecommunications and Multimedia, Helsinki University of Technology, Espoo, Finland, 1999, 22p.

[Laakso 2001]

Laakso, M. (2001). *Practical Navigation in Virtual Architectural Environments*,

Master's Thesis, Laboratory of Telecommunications and Multimedia, Helsinki University of Technology, Espoo, Finland, 2001.

[Lipton 1997]

Lipton, L. (1997). *StereoGraphics Developers' Handbook - Background on Creating Images for CrystalEyes and SimulEyes*, StereoGraphics Corporation, 1997. 65p. Also available on-line at <URL:http://www.stereographics.com/>

[Loughlin and Hughes 1994]

Loughlin, M.M., Hughes, J.F. (1994). *An Annotation System for 3D Fluid Flow Visualization*, Proceedings of Visualization'94, Washington, DC, Oct. 1994, pp. 273–279.

[Napari 1999a]

Napari, H. (1999). *CAVE-visuaalijärjestelmä*, Special Assignment in Telecommunications, Department of Electrical and Communications Engineering, Helsinki University of Technology, Espoo, Finland, 1999, 40p. (in Finnish)

[Napari 1999b]

Napari, H. (1999). *Magic Lens user interface in virtual reality*, Master's Thesis, Laboratory of Telecommunications and Multimedia, Helsinki University of Technology, Espoo, Finland, 1999.

[Post and van Wijk 1994]

Post, F.H., van Wijk, J.J., (1994). *Visual Representation of Vector Fields: Recent Developments and Research Directions*, in Scientific Visualization, Advances and Challenges, Rosenblum, L., Earnshaw, R.A., Encarnação, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D., Editors, London, UK, 1994, pp. 367–390.

[Rajlich 1998]

Rajlich, P. T. (1998). *An Object Oriented Approach to Developing Visualization Tools Portable Across Desktop and Virtual Environments*, Master's Thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1998. Also available on-line at http://brighton.ncsa.uiuc.edu/~prajlich/T/

[Ruokolainen and Gröhn 1996]

Ruokolainen, J., Gröhn, M. (1996). *Tieteellinen visualisointi*, CSC - Tieteellinen laskenta Oy, Espoo, Finland, 1996.

[Sadarjoen et al. 1998]

Sadarjoen, I.A., de Boer, A.J., Post, F.H., Mynett, A.E. (1998). *Paricle Tracing*

*in Sigma-Transformed Grids using Tetrahedral 6-Decompsition*, Proceedings of the Eurographics Workshop, Visualization in Scientific Computing '98, Blaudeuren, Germany, April 20-22, 1998, pp. 71–80.

[Savioja 1999]

Savioja, L. (1999). *Modeling Techniques for Virtual Acoustics*, Doctoral Thesis, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, Report TML-A3, Espoo, Finland, 1999. Also available on-line at `http://www.tml.hut.fi/Research/DIVA/publications.html`

[Schroeder 1999]

Schroeder, W. J. and Martin, K. M. (1999). *The vtk User's Guide*, Kitware, Inc., 1999.

[Shulz et al. 1999]

Schultz, M., Reck, F., Bartelheimer, W., Ertl, T. (1999). *Interactive Visualization of Fluid Dynamics Simulations in Locally Refined Cartesian Grids*, IEEE Visualization'99 Proceedings, 1999, pp. 413–416.

[Seiro 1998]

Seiro, E. (1998). *Video photography course*, Lecture given by cinematographer Erkki Seiro, lecture notes, Helsinki, 1998.

[Tramberend 1999]

Tramberend, H. (1999). *Avocado: A Distributed Virtual Reality Framework*, IEEE Virtual Reality, 13-17.3.1999, Houston, Texas, USA, 1999.

[Woo 1998]

Woo, M., Neider, J. and Davis, T. (1998). *OpenGL Programming Guide: the official guide to learning OpenGL, version 1.1*, 2nd ed. Addison Wesley, USA, 1998.

[Wright 2000]

Wright, D. (2000). *A Survey of Projection-Based Immersive Displays*, in Displays and Virtual Reality Systems VII, John O. Merritt, Stephen A. Benton, Andrew J. Woods, Mark T. Bolas, Editors, Proceedings of SPIE Vol. 3957, pp. 482–492 (2000).

[Äyräväinen 2000]

Äyräväinen, S. (2000). *Keinotodellisuuden soveltamismahdollisuudet sotilaskoulutuksessa*, Master's Thesis, Laboratory of Telecommunications and Multimedia, Helsinki University of Technology, Espoo, Finland, 2000.