Interactive visualization of flowfields in an Immersive Virtual Environment

Applied to the test case of simulated abdominal vascular reconstruction

Author:	J.M. Ragas
Study:	Computer science
Department:	Section Computational Science
University:	University of Amsterdam
Supervisor:	drs. R.G. Belleman
Professor:	prof. dr. P.M.A. Sloot
Date:	August 2002

J.M. Ragas
jmragas@science.uva.nl
Faculty of Science
Section Computational Science
University of Amsterdam

Cover pictures
The river Binn crosses Binntal, Swiss, at an altitude of 1400m. The two pictures are taken with a shutterspeed of respectively $\frac{1}{800}s$ (left) and $\frac{1}{4}s$ (right), with a period of approximately 10s between the two snapshots. The pictures serve as a good example of accurate though completely different representations of the same flow. Neither visualization is "better" or "worse" in its representation.

Abstract

The analysis of multi-dimensional flow data demands a scientific representation environment which supports an effective study by a scientist or expert, in order to reduce design and production costs and aid in making critical decisions. The visual representations must clarify mutual relations, time-dependencies, location and scale of interesting phenomena. To support effective interactive analyses of these often large and complex flow domains, scientific flow visualization methods in virtual environments are used. The presence of a human in the visualization as well as the 3 dimensional nature of the environment and visual representations demand intuitive interaction and navigation possibilities.

In this thesis we present FlowFish: a visualization library aimed at visualizing flows in an immersive virtual environment. It includes several visualization methods, each capable of presenting a unique representation of flow properties to the user. Interaction with the visual representations is performed by means of 3D widgets such as cursors and menus. A test case is provided in which FlowFish is used to examine and analyze medical data, consisting of simulated bloodflow through the human aorta.

The use of interactive virtual environments imposes constraints on the performance of the used visualization methods. Difficulties arise when qualitative and performance goals conflict. Several options are implemented to reduce the complexity while keeping a good overview of the flow's behaviour, but the end-user will always have to decide whether and to what extend he prefers either performance or quality.

FlowFish shows that flow visualization in an immersive environment has great potential by offering high quality representations. However, the parameterization of visualizations remains a problem. Numerous parameters must be set before a good visual representation can be achieved while indications of useful settings are seldom available.

keywords: flow visualization, immersive virtual environment, CAVE, glyph, streamline, particle animation, raycasting, iso-surface.

Acknowledgements

Things are going to slide, slide in all directions,
Won't be nothing,
Nothing you can measure anymore.
Leonard Cohen - "The Future"

After all the hours I spent on this thesis, it feels good to relax the language muscles and let them slip aside for a moment from the small strict road to which they were confined during the scientific writing such a document requires. I do take this opportunity of expressing my gratitude to some of the people who supported me during the two years it took to complete this thesis. To all the others I do not name specifically (you know who you are), thank you.

I thank Peter Sloot for providing this opportunity to me. Visualization in an immersive environment has been, and still is, a mighty interesting subject, and I cannot think of any other way I could have performed such a research. To allow me to continue the research after this thesis is finished, I do consider a privilege. I would like to add I would never have guessed that the group leader of the Section Computational Science would vote KinkFM the best radio station of the Netherlands.

My immense gratitude goes to Joost, first among equals. With whom I raised spam to an art, laughed at the world, discussed and listened; who sometimes kept me sane, and other times pushed me through insanity to come out on the other side and made me able to continue. You're the best.

My parents, sisters and Linda. Who all made a point of *not* asking me how it was going, realizing other people did already ask often enough. Your support has been invaluable. Henny, with whom I shared the room, the experience of conquering Europe, making countless cups of espresso and lots of music. Let us never cease reaching for a 147.

A special note goes to Arjen of KinkFM, for showing me a world beyond science, consisting of wonderful music, poetry, literature and all kinds of discussions at hours most people wouldn't even dream of being awake. I would never have grasped the true meaning of *in vino veritas* without you.

Then, finally, Rob. Without you, this would not have been possible. The extensive knowledge, both theoretical and technical, the guidance, the patience with my stupid little pranks, and taking me serious at other moments; all that and more made this possible. I enjoyed the discussions we had during the last months, concerning visualizations and their (im)possibilities, which sometimes extended far beyond the subject of this thesis. Your countless scrutinized reviews improved both my way of thinking (concerning science, that is) as well as my writing technique. Thank you very, very much.

Contents

Chapter 1

Introduction

The computer has become an invaluable tool in scientific research. The increase in computing power and memory-capacity of today's computer systems has resulted in an increase in both complexity and size of the data spaces generated in many scientific areas [1]. Scanning devices that acquire a digital representation of real world phenomena show the same trend in increasing performance. As a result of this increase, computational scientists have been able to perform increasingly large and more complex computer simulations.

The accuracy of computer simulations partially depends on the resolution and size of the data. A higher degree of detail results in a more accurate description. Obtaining higher resolution was prohibitively costly, because of supercomputer prices and the time needed to solve equations. Scientific research benefits from the increase in computing power and data storage as simulations can handle larger datasets with increased resolution [2]. The level of detail is increased while the total computing time as well as the error in the results are reduced. Still, the need for more accurate results will not diminish. Because datasets keep growing, the methods which help to analyze them have to be adapted as well.

1.1 Data analysis

The results of scientific simulations frequently contain high-dimensional data which can often not be analyzed analytically or numerically, since there are no well-defined algorithms that extract the desired information from the raw data. Even simple systems show complex behaviour which have no known analytical solution [3]. In these cases, other means of data-analysis are needed to gain insight in the data.

The capacity of computers to automatically explore data and find interesting phenomena, is limited. Computers lack the cognitive properties of the human mind. In most cases it is very difficult to capture the definition of "interesting patterns" in an algorithm. Often the exact nature of the phenomena is not even known beforehand. In these cases the researcher usually resorts to other means of analysis, such as, for example, scientific visualization. By bringing the "human in the loop", the best aspects of computer simulation and intuitive real-world interaction are combined. The goal of representing data to the scientist

is to maximize research efficiency and minimize research time.

1.2 Data presentation

The human mind can only comprehend a limited amount of data at the same time, therefore the amount of presented information must be reduced without removing areas of interest. The creation of a simplified representation is not self-evident. The factor by which scientific data can be simplified without omitting important information, is an inherent property of the data.

A useful representation presents as much information as possible by presenting as little data as possible. To meet these demands, every scientific area requires its own unique methods which implies that the parameters required to control the representation are not constant. For every unique dataset they will have to be defined separately. Since the analysis of the data cannot be performed automatically, the researcher must explore the mapping of scientific data onto representation parameters.

1.2.1 Data presentation modalities

There are various methods to present data, each with its benefits and shortcomings. Scientific visualization is the most common method. Other modalities can be used, such as sound, touch or smell. A representation must be capable of handling and presenting the available amount of data, its granularity and dimensionality. Global as well as detailed representation methods must be available.

Tactile or force-feedback techniques are powerful ways to provide information. By touching an object, properties such as shape, texture, temperature and density can be conveyed to the scientist simultaneously. A solid object can be simulated by using a so-called data-glove [4]. By wearing this glove, a scientist can touch and feel an object which exists only in computer memory. The current physical properties of the glove are still limited. The detail in texture is coarse and the realistic imitation of density is difficult to accomplish. Their lack of constraints make them difficult to use.

Sound is a useful method of data representation. The human ear can simultaneously distinguish multiple frequencies and timbres, as well as their rhythm and volume [5]. Effects such as echo and reverb can help to position a source of sound in 3D. A researcher can quickly recognize a particular tone, instrument or the overall melody. Though audio might not always be useful to represent a complex dataset, it can be helpful for smaller subsets or a single object. It does not require physical orientation of the scientist nor special or advanced equipment.

As the old verb states, a good picture is one that speaks a 1000 words, and since many years the visual representations of scientific problems have facilitated complex research [6]. Confronted with a large amount of information, visual exploration methods are essential to analyze the information or target the problem present in the data.

While focusing on a specific property or area, other parts of the data will be disregarded. The absence of visualized data can be deceiving when the visualization is not tuned with the right parameters or puts emphasis on the “wrong”

detail. Displaying a large amount of data can clutter the view, thereby potentially obscuring features that are of interest. It is not trivial to show as much information as possible by showing as little data as possible. A compromise between the amount of data and the amount of information must be found. This is the subject of scientific visualization.

1.3 Interactively guided scientific Virtual Reality environments

Images are powerful and can present a lot of data at the same time, but this is not necessarily the information the scientist is looking for. To get a useful visual representation, different settings and approaches must be tried which requires constant interaction. A scientific visualization tool remains just a tool: it is the scientist who directs the orchestra of visualization methods and who performs the final parameter-tuning.

One generic visualization tool suited for a large range of different research areas is not preferred. It would have to include a large amount of different representation methods because the properties of data differ considerably between research areas. The result is a large, complex library which includes tools for every possibly type of data. This is not a very practical approach. For the visualization of a specific type of data, a large amount of the included tools would never be used. Libraries with subject-dependent representation methods avoid this problem and the optimization of specific methods is easier. Only visualization tools suitable for the specific data are presented to the researcher. The environment must offer interaction methods to change the representation and its parameters. Since many datasets are large and complex, easy navigation is necessary to efficiently study the data without spending too much time moving to the desired viewpoint.

1.3.1 Virtual Environments

On average workstations, the hardware needed to create advanced visualizations is often already available for an average price. The support for these graphics cards by common software standards such as OpenGL [7] has increased their popularity.

Applications benefit from this increased performance by presenting more complex scientific visualizations. A 2D screen severely limits the possibilities; large amounts of displayed objects quickly clutter the view. 3D objects can be drawn by using perspective projection which provides an additional dimension, allowing more information to be presented [8]. But navigation and interaction is difficult, consuming valuable research-time.

An immersive virtual environment provides advanced methods for visualization. The idea is to fool the senses in such a way as to make the artificially created world look and “feel” real. The “immersion” of the observer in the data provides a richer experience which facilitates the study of complex data structures in less time, compared to desktop visualization environments. Perceiving the virtual environment as a 3D space all around him, the user resides at the very center of the visualization. The result is a perceptual realistic and intuitive environment, resembling the way people observe and interact with the

real world. In the field of complex scientific visualization, the benefits of such an environment are multitude. The major impact of virtual reality (VR) technology on scientific visualization is in providing a real-time intuitive interface for the exploration of data, facilitating the use of scientific visualization in the research process [9].

1.4 Goal of this research

This thesis describes a flow visualization library, FlowFish, which creates flowfield representations in an immersive virtual environment. All flowfields have certain characteristics in common. A description of the necessity of these tools as well as their functionality in flowfield research will be given, including advantages and disadvantages of the individual methods.

The presence of a human in the process of analyzing data poses requirements on the functionality of the library. The overall response time must be kept as short as possible. A constant long delay results in a visualization which most of the time is not synchronized with the user's actions. As a consequence, efficiency is reduced during interaction.

The FlowFish library was designed to be a set of tools which provides an intuitive environment suitable for a detailed analysis of flowfield properties. As a test case, a medical application depicting blood flow through arterial structures is used.

1.4.1 Chapter overview

A general background on flowfields, their properties and characteristics, is given in chapter 2. It describes methods of analysis and the shortcomings of desktop visualization methods. Chapter 3 describes the FlowFish library, its functionality and visualization methods, their use, implementation and results. The medical test case is described in chapter 4, followed by conclusions and future work in chapter 5.

Chapter 2

Flowfields

Gasses and fluids are present everywhere. From blood flowing through small veins to the massive movement of the tide, flows play a mayor part in many research areas. Diversity in origin and composition is immense, but the basic properties are the same and the same physical laws apply to them. A better and detailed understanding of flows, their physics and behaviour, will lead to more effective instruments and research. To create an exploration environment suitable for scientific flow examination, the properties of the data in relation to this environment must be considered. Amongst these properties are the following:

- Data properties. Which properties are available in the data? What do they represent? Which of these properties are important for the research? How do they relate to one another? Can some of them be omitted? How do these properties change over time? Do these changes occur on a global or local scale? Can the different properties be compared?
- Environment properties. What are the specifications of the environment? What kind of visual representation methods are supported and are suitable to represent the data? Which visualization and exploration demands must be met? What type of navigation is required? What kind of interaction methods should be available?

Flowfields appear in such a wide variety of applications and conditions that when they are the subject of scientific research, there is not a single or best approach. The goals of different research areas will be aimed towards different macroscopic or microscopic properties of the flow. Some general interesting or important characteristics always apply though, such as direction and speed. These are basic properties of the flow that can not be omitted.

A visualization library must try to be as applicable to any flow and research goal as possible, since there is not a single description of conditions or set of equations describing the interesting phenomena present in a flow. A more abstract representation allows many approaches and could support different research areas.

Basic properties of flow are the following:

- Direction. The “global” direction of the flow, that is, the imaginary line(s) a flow follows when observed by a human eye, is caused by the convergence

of small local flows within the flow. The billions of flow-particles each travel in their own direction with their own speed. If locally most of the particles agree on the same direction and speed, the flow appears to behave smoothly, without big disturbances. Particles might diverge or converse, due to small changes in speed, direction or geometry within the flow, causing phenomena such as rotational- and back-flows.

- Pressure. The pressure working on a flow consists of both the inner pressure of the flow itself as well as pressure working on the flow from the outside and the effects of the flow-pressure on the surrounding boundary or structure.
- Wall shear stress. By moving alongside a boundary, a (semi-)solid structure or flows with different properties, a flow generates a "drag force" on the boundary: this so called shear stress is directly related to the gradient along the streamwise direction. Both force and direction are important when considering the effect of the flow on solid structures.
- Viscosity and temperature. Viscosity is a measure of a fluid's resistance to flow. Often, a fluid's viscosity depends on its temperature. Depending on the research goal, these properties can be of interest.
- Dynamics. Many flows show dynamic behaviour over time. Instead of a single snapshot of the flow, a dynamic representation may be necessary.

2.1 Flow simulation

Many flow studies can be conducted in research laboratories. Flow instruments can quantify the properties that are of interest. However, an unwanted effect during flow measuring is the possible influence of the used measurement techniques on the flow itself. For example, sensors placed inside a flow can directly change its properties. In other flow research environments, such as flow in the human vascular system, measurements are not always possible due to potential health risks.

During the mechanical engineering of objects which are in direct contact with a flow (such as boats or oil pipelines) a lot of tests have to be performed in order to get an efficient design. Test environments such as wind tunnels or water basins can be used for verification and testing purposes, but the number of conditions and configurations that would potentially require testing would be impractical with the exclusive use of experiments. The detail of each prototype configuration would require an enormous amount of preparation time while engineers are limited in evaluating design modifications by high cost and available time of physical testing as well as scalability concerns. Simulations can be used to correct implementation errors in an early stage.

Flow simulation potentially provides methods to both lower the costs associated with design and enable the investigation of flow non-invasively. The results of a simulation can be used to verify a hypothesis or theoretical model. It attempts to prevent over-designing by eliminating preliminary models at the start of the design process and brings once-elusive problems under a measure of control. Behaviour of flow in previously inaccessible places can be observed and investigated.

Simulation is not a substitute for real testing, but rather a useful tool that, once validated, can reproduce conditions that would be impossible or impractical to duplicate in physical testing [10].

Example applications of flow research in industry.

Shell Research has used discrete simulation methods to compute viscous flows and performance measurements of pipelines and flow separators [11]. Single phase and condensing heat transfer in any geometry (e.g. a pipeline) used in the process and power plant industries are simulated. The goal is to assess and validate the current methods and perform simulations on shell and tube heat exchanger designs.

Secondary oil recovery is a process that is applied in the majority of oil reservoirs in the North Sea. Water is injected into the reservoir to maintain reservoir pressure (thus preventing excess gas production) and to displace oil towards the production wells. The efficiency of the displacement is controlled by the reservoir rock permeability, the fluid viscosities and the relative permeability of water to oil and oil to water. Sintef, one of Norway's leading independent research groups [12], has used flow simulation to increase production in a North Sea oil field by helping to conceive and validate design decisions [13]. Computational Fluid Dynamics were used to analyze velocity, pressure and concentration of multi-phase fluid flow within the separator, which previously was the limiting element in the production process. A simulation was run which showed that in the original design some of the internal parts of the separator caused rotational flow structures which produced backflows thereby reducing efficiency.

2.1.1 Modeling flow

The behaviour and movements of a flow are often too complex to be described by analytical or numerical methods. Discrete equations in which the flow is modeled by a finite number of particles, are often inevitable.

The flow is characterized by the Reynolds number which is defined as:

$$Re = \frac{v * d}{\nu} \tag{2.1}$$

where v is the (either mean or maximum) velocity (mm/sec), d the diameter at a chosen plane (mm) and ν is the kinematic viscosity (mm^2/sec). To define a stable flow, the Reynolds number suffices. The Womersley parameter α has more influence than the Reynolds number in the nature of unsteady flows [14]. It is defined as.

$$\alpha = R\sqrt{\frac{\omega}{\nu}} \tag{2.2}$$

where R is the radius of the tube, ω the angular frequency and ν the kinematic viscosity.

There are several methods available to model flow for the purpose of flow simulation, each varying in complexity, accuracy and applicability. While choosing a method, a scientist must balance size, dimensionality and detail of the flow he wants to simulate against the computing power he has at his disposal. Even with small flows, calculations can take days or weeks to finish. There is no "perfect choice" when deciding upon the method and simulation parameters. The capabilities of hardware keep expanding but the demand for better and increasingly detailed simulations as well. Because of practical limitations, concessions will always have to be made. An immensely detailed simulation might be very realistic, but it is useless when it takes years to complete.

Computational Fluid Dynamics (CFD) includes various methods to compute viscous, possible turbulent flows. Studies indicate a fairly high level of parallelism on clusters of commodity machines making solutions for practical problems affordable [15]. Problems such as turbulent flow prediction can be solved by, for example, implementations such as thin-layer, coupled thin-layer and parabolized Navier-Stokes code [16]. Domain scaling is not an issue with CFD because the simulation can easily be performed at the actual size [13].

Examples of flow simulation methods are the following:

Molecular Dynamics (MD). MD uses classical models of atomic interactions simplified from quantum mechanics. It helps in understanding the structure and function of biomolecules such as proteins and DNA which is crucial to the understanding of mechanisms of drugs and life processes [17]. Advanced MD simulations can lead, for example, to better treatment of diseases. A system is modeled as a collection of molecules consisting of atoms connected by bonds and interacting via electrostatic and van der Waal's forces. Time steps of typically one femtosecond (10^{-15}) must be used due to high frequency bond vibrations, though many phenomena of interest occur on time scales of nanoseconds (10^{-9}) or longer. The force determination from which the potential field is computed consumes most of the runtime, often 90% or more.

Finite Element Method (FEM). FEM is mainly used in the area of electromagnetic problems, such as eigenvalue problems [18]. It is capable of simulating arbitrary shaped objects filled with complex materials or the aeroelastic deformation of objects [19]. FEM is based on a numerical solution of a macroscopic description of fluid flow. It uses variational calculus which allows the transformation of a set of partial differential equations (e.g. the Navier-Stokes equations) into a system of linear algebraic equations.

Lattice-Boltzmann (LB). LB is a mesoscopic approach. It represents a method for solving the Boltzmann Equation (in its original form). It originated from the lattice-gas model [20]. The key idea behind the Lattice-Boltzmann method is to model fluid flow by distributions of particles moving on a regular lattice. At each time step the particles propagate to a neighbouring lattice point followed by local collisions in which velocities are redistributed. An important advantage is the inherent spatial locality of the updating rules. This makes it ideal for parallel processing [21].

2.2 Flow analysis

2.2.1 Characteristics

In science, three states of matter are recognized: solid, liquid and gas. Liquids and gas are both fluid. In contrast to solids, they lack the ability to resist deformation. Because of this inability to resist, fluid moves. The deformation is caused by shearing forces which act tangentially to a surface. The shape of the fluid changes continuously as long as the force is applied.

A flow consists of millions of molecules, all moving and pushing against each other. Within small areas, all particles are generally moving in the same direction. Basic properties of the flow such as speed, direction and pressure all depend on the properties of the individual particles and the structure of the surrounding environment. Directly related to the movement is the speed of the flow. A relatively high, low or fast change in speed will often indicate an interesting area. Speed and direction create pressure which can deform a structure through or around which the flow moves.

Many flows are time-dependent. Changes are recorded by taking several snapshots of the flow, separated by a constant time-period. By putting these snapshots in sequence, the behaviour of flow over time is reconstructed.

2.2.2 Automatic feature extraction

The interesting features of flow from a scientific point of view are phenomena such as vortices (rotational patterns), singularities, boundary layers (transition of laminar to turbulent flow) and recirculation zones (rotational backflow)[22]. To analyze the behaviour of a flow these phenomena must be located. On a small scale, many of these features can be present. The objective of flow analysis is to filter out and present the important macroscopic phenomena.

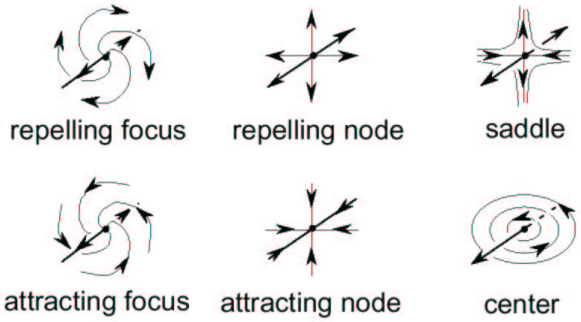


Figure 2.1: Critical point classifications (from [22]).

Automatic feature extraction is a method to automatically deduce the location, shape and strength of specific features without human intervention. Vector field topology is a feature extracting algorithm suitable for analyzing flow on a basic level. The concept behind vector field topology is to divide flow into closed regions in which the local behaviour is “similar”. The algorithm determines the critical points (where velocity = 0) in the flow. From these critical points, virtual particles are integrated along the eigenvectors. By connecting

their subsequent positions, so-called streamlines are created. These are used to classify the critical points (figure 2.1).

At present, there are no robust computational methods to extract a complete set of singular streamlines and stream surfaces from numerical simulations of fluid flows [23] which limits the possibilities of automatic analysis. Furthermore, it is very difficult to describe an algorithm which extracts useful information from these classifications. In related work a hybrid form of critical point classification and visualization is applied for the investigation of flowfields [22].

2.2.3 Flow visualization

In cases where the complexity of flow is too high to extract the required information automatically, the researcher must resort to other methods such as manual analysis. The goal of flow visualization is the creation of an environment in which a flow representation facilitates a fast and correct analysis. Flow data consists of both scalar and vector data which are mapped to properties of visual objects, such as location, orientation, size, colour and transparency. Interesting phenomena in the flow are often positioned on a specific data domain. They are highlighted by mapping this domain on a unique value or subdomain of a visual property.

The general direction of a flow can not be measured with a single vector. It may seem to move in several directions, or perhaps nowhere at all. To examine the fluid's direction, tools must visualize the direction on many locations within the flow. Together they provide the researcher with an impression of the general direction(s).

The visualization of vector data such as flow direction or shear stress creates a visual representation by aligning visual objects to flow data. Similar data vectors result in similarly oriented objects, enabling a quick overview of the flow, spotting of areas containing similarly aligned vectors and fast localization of diverging vectors. The visual representation of speed is often combined with the representation of direction. Performing a mapping from minimum to maximum speed onto a visual property such as colour or transparency highlights possible interesting areas. The visualization of similar scalar flow properties such as pressure, temperature or viscosity requires a likewise mapping of minimum to maximum values (either implicitly or explicitly presented in the data) to a visualization property. Several methods of mapping scalar data onto visual properties must be present. Linear dependencies in the flow data should result in perceivable linear dependencies in visual properties.

Texture methods such as the spot noise algorithm [24], line integral convolution (LIC) [25] and hypertextures provide visualizations of dense flowfields as 2D or 3D textures, but are computationally expensive. They use random spots which are rotated, scaled and advected by the local flow, or a flowfield is used to filter an input texture.

Visualization methods must be able to deal with time-dependent flow data. When a subsequent snapshot of the flow is loaded the visualization methods must update themselves. When lookup tables are used, a choice must be made whether their mapping-range is updated as well: keeping the range the same maps the same values to the same colour or transparency value. But the new flow snapshot can contain extended scalar ranges which fall outside the lookup table's range and will therefore be mapped to the same colour or transparency.

Updating the table's range will prevent this, but the same scalar values will not be mapped to the same table-values.

2.3 Scientific visualization

Scientific visualization is a powerful way to present complex 3D data. It is oriented towards the informative display of abstract quantities, relations and concepts, as opposed to attempting to realistically represent objects in the real world. The graphical demands of scientific visualization are oriented towards accurate, as opposed to visually realistic, representations.

Visualization of scientific data is not self-evident. Detailed data is necessary for successful representation, but it must not result in a chaotic scene. Adequate and powerful visualization tools must prevent cluttering while not obscuring interesting aspects of the data.

Common visualization environments are available on many kinds of workstations, each with its own strengths and weaknesses. Different visual representations can be chosen, ranging from simple 2D schemes such as histograms and diagrams to advanced 3D pictures. Present-day workstations are often equipped with fast CPU's and high-speed graphic-cards, enabling high performance scientific visualization without the need for expensive dedicated hardware [26].

Examples of industrial scientific visualization applications

IRIS Explorer is a visual programming environment for 3D data visualization, animation and manipulation [27]. It was originally developed by Silicon Graphics, Inc [28] but the development was transferred a few years later to the Numerical Algorithms Group [27]. IRIS Explorer works on multiple platforms and uses several graphics libraries. Its use is not limited to the interactive analysis of data but it also presents an Application Programmer's Interface (API) for the construction of specific routines.

Advanced Visual Systems' Express offers advanced visualization support including vector field stream ribbons, jigsaw cell face extrusion and two-pass transparency [29]. It contains more than 850 visualization objects matching the most common application areas such as CFD, Finite Element Analysis (FEA), Medical, Geographic, images and geometric models. It includes interactive functionality and can convert output to streaming video formats such as AVI, MPEG and VRML.

IBM's Visualization Data Explorer provides possibilities ranging from simple dataset visualizations to the examination and analysis of complex time-dependent data [30]. It includes a fully operational graphical user interface (GUI) for the interaction between user and data. OpenDX is the open source software version of the Visualization Data Explorer. Its main goal is to offer powerful features for free, added by independent visualization developers.

VTK is Kitware's open source system for 3D scientific visualization and image processing [31]. It supports multiple interface layers and a wide variety

of visualization algorithms including scalar, vector, tensor, texture and volumetric methods. The object oriented class library contains advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing and Delaunay triangulation. As an open source project, every developer can improve or extend the library.

2.3.1 Scientific visualization in Virtual Environments

Flow data often consists of high-dimensional data in a 3 dimensional volume. 2D representation limits the amount of available information because only a "slice" of the 3D volume can be visualized. The representation of the 3D spatial relationship can not be correctly performed in 2D and, inevitably, loss of information occurs. Furthermore, navigation and orientation of a scientific visualization on a 2D screen can be tedious and time-consuming as interaction by means of a mouse, trackball and keyboard are limited. An environment in which the researcher is truly immersed in the visualization offers a more detailed and intuitive visualization of complex flow data while allowing every orientation and navigation possibility the user wants.

Immersive virtual environments offer a "true" 3 dimensional view of the scientific visualization. The user is immersed in the environment and the visual representations are created all around him. Motion-parallax can be used to get a good depth-cue of visual representations: by looking at objects from slightly different angles, a quick estimation of size and distance can be made. It is an intuitive method for orientation in 3 dimensions and is therefore an important advantage of 3D VR systems over common 2D visualization.

The immersive environment in which the visualizations are performed must present maximum freedom during exploration. The presence of a human in the analysis process demands the exploration environment to be user friendly and easy to control. Navigation and the graphical user interface (GUI) must be intuitive so that no extensive training is necessary. A rich set of spatial and depth cues should be provided, facilitating easy orientation of the researcher in the immersive environment as well as a good estimation of size and location of visual representations. The environment offers 6 degrees of freedom (DOF), which, compared to common 2D scientific environments, allows a larger freedom of movement and interaction. It must provide tools with which the data properties and size can be measured so it can be compared with real-life examples [32].

The CAVE

Carolina Cruz-Neira et al. define virtual reality as "a system which provides real-time viewer-centered head-tracking perspective with a large angle of view, interactive control, and binocular display" [33]. They focus primarily on the visual aspect of virtual environments when they describe their CAVE Automatic Virtual Environment (CAVE) [34]. The definition of general VEs often includes a broad description of other human sensor in- and output.

The primary motivation of the CAVE design (see figure 2.2) was to create a useful tool for scientific visualization. To be able to successfully compete with

Figure 2.2: The CAVE at SARA.

existing (stereo) visualization tools, the design must be able to meet the expectations of scientists who are used to the high-resolution graphics of common workstation visualization software. Distinctive features of the CAVE include:

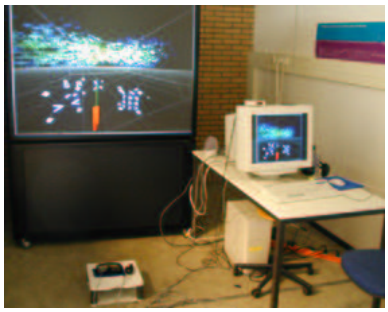
1. good surround vision without geometric distortion,
2. less sensitivity to head-rotation induced errors,
3. the ability to mix VR imagery with real devices,
4. the ability to couple networked supercomputers and data sources.

The CAVE installed at SARA in 1997 consists of 3 10x10 ft. rear-projection screens for walls and a down-projection screen for the floor (figure 2.2). With a frequency of 120Hz, full-colour 1024x768 images are displayed, providing each eye with a full-coloured stereoscopic 60Hz display. Stereo-Graphics LCD shutter glasses, synchronized with the projection screens, are used to separate left-eye from right-eye images. The user's head and hand position and orientation are tracked using 6 degrees of freedom (DOF) magnetic trackers while a "wand" is used for interaction with the virtual environment. Audio input and feedback is provided through a wireless microphone and stereo audio-system. The computing platform is a SGI Onyx2 with 8 R10000 processors running at 195MHz, each with 128Mb ccNUMA (cache-coherent Non-Uniform Memory Access) memory for a total of 1,024Mb and 4 InfiniteReality2 graphics pipes, each with 2 RM7 raster managers.

Introduced at the SIGGRAPH conference in 1992, the abilities of the CAVE have been expanding ever since. The implementation of the CAVE at SARA [35], at which some of the early testing of FlowFish (see chapter 3) was done, resembles much from the original design, both in hard- and software. The graphics library used by the CAVE is based on the OpenGL standard. To facilitate high performance visualization, hardware-support for OpenGL is a necessity.

The most important way of interaction in the CAVE is by means of the wand. This device acts like a 3D mouse with 6 degrees of freedom. Navigating in a VE requires changing the researcher's position, as opposed to changing the location and orientation of the environment which is common on 2D VEs.

The UvA-DRIVE system



a: DRIVE workstation (bottom-right) with projection screen.

b: Projection screen, shutter-glasses and wand.

Figure 2.3: The UvA-DRIVE system.

The Distributed Real-time Interactive Virtual Environment (DRIVE) system designed at the Computer Science Institute of the University of Amsterdam differs both in hardware and software (figure 2.3) [26]. A 3,3' x 5' rear-projection screen is attached to a parallel processor (2 x 1GHz) workstation with 1Gb of memory, running Red Hat's version of Linux. Head and wand-tracking are done by a Polhemus Fastrak tracking system [36] and audio input is provided by cordless microphone and receiver. Compared to SARA's CAVE, it can facilitate fewer users at the same time (approx. 5) but this setup provides a fast, high-quality, easily upgradable VR system for only a fraction of the cost.

The DRIVE is equipped with an ASUS V7700 Deluxe GeForce2 Ultra, 64Mb DDR Ultra. It has a 1Gpixel/sec fill rate, 31M triangle rate (triangle/sec), AGP 4x and supports a maximum resolution of 2048 x 1536 at 75 Hz.

By using a resolution of 1024x1576 at 67.5Hz, images for the left and right eye are displayed simultaneously at a resolution of 1024x768, separated by 40 black scanlines. A StereoGraphics device takes care of splitting the image into 2 separate images, displaying them alternatingly on the projector. The shutter glasses are synchronized to the alternating images by means of infra-red lightpulses.

2.3.2 Design issues of a VE application

In contrast to conventional 2D graphics environments a VR environment is usually not a completely event-driven run-time architecture [37]. These architectures assume that user actions are the primary driver of actions in the environment. They are idle as long as no event-input is provided.

The environment is expected to provide a reliable and consistent representation of the results of the simulation at that moment and mechanisms enabling the user to change parameters in the environment. All these actions are events which must be processed seemingly parallel by the application [38].

A loop architecture which is constantly executing commands in an "infinite" loop provides therefore an easier approach. The disadvantage of this approach is a strong dependency between otherwise independent modules: The total execution time and speed of the loop is limited by the time to perform the slowest

operation.

The main actions performed in the loop are:

- Visualize data. The data is loaded and converted into a visual representation by varying methods. The time needed to perform these operations depends on the size and complexity of the data, the chosen visualization methods, their algorithms and parameters.
- Interaction interface. The researcher navigates and interacts with the system to adjust parameters. The notifications of interaction with the system, such as a pressed button, are called events. These must be passed on to the appropriate objects. The events themselves do not take much time to process, but the visible results must not be delayed too long, or the researcher will experience difficulty in the interaction process.
- Rendering images. The final visualization of data as well as graphical user interface (GUI) items such as buttons must be rendered to the screen. The speed is limited by hardware performance such as the vertex throughput and the vertex processing time.

2.3.3 Optimization

The time between an action of the researcher and the resulting visual feedback is called the interaction response [39]. If the environment reacts too slow, the user will experience difficulty in navigation and during the precise placement of tools. Often, a lot of small adjustments are needed to find the correct representation. If the lag between subsequent adjustments is too big, it will limit effectiveness and extend research time considerably, decreasing the benefits of the virtual exploration environment.

The interaction response depends on the transformation of data into visual objects and the render time needed to draw these objects:

- The transformation of scientific data into visual objects and their properties (colour, transparency) must be fast, though the allowed period is longer than the allowed render time, due to its infrequent occurrence. The time needed to update the visualization should be kept low without compromising consistency. Transferred data and visualization computations must be minimized as much as possible. This can be accomplished by only recomputing changed parts of the data-transformation process [40].
- The overall render time of the visualization environment must be sufficiently low. The render time is the time needed to convert triangles and polygons into screen-pixels. When the render time is too high, navigation and interaction will become difficult since the visual feedback occurs too slow to allow interaction. Usually a framerate of at least 10 frames per second is necessary to allow smooth exploration and interaction. To accomplish this framerate, visualizations might be simplified by reducing the amount of polygons or by visualizing a subdomain of the flow.

To control the interactive response and the speed of visualization updates, the amount of visualized data must be limited as much as possible. It depends on the data and the research goals which methods are suitable to improve the

speed of visual feedback: in some cases a researcher might decide a slow visual update and detailed visualization is preferable over a fast updated and less detailed view.

Object optimization

An object can examine its input data, as well as its own parameters. In order to improve performance it can choose to change its state by using this information. No feedback is provided to compute whether the change in state had the desired effect on performance improvement.

One of the easiest methods to compute the speed of the visualization application is by using the current number of frames per second (fps). This method has a number of drawbacks though:

- The period between subsequent updates of the object's state is hard to guess. An object which changes its parameters constantly can create a confusing visualization. If the object reacts too slowly to user- or system events, adjustments are applied to the object whose actual parameters and state are already advanced but not yet visualized, which results in an inconsistent state of the environment and confusing feedback towards the user [41].
- The number of frames per second does not depend on one object only. When a low framerate occurs, several objects might start to adjust themselves at the same time. It might be just one object which causes the number of polygons to drop dramatically. To circumvent this problem, one could use the object's amount of triangles or polygons, as opposed to the total amount of polygons in the scene. The visualization object itself however, is not responsible for automatically adjusting its parameters when the framerate drops below a certain threshold. It is the responsibility of the render environment to decide which object should adjust itself: this is the only part of the visualization application which can compare all the objects and their relative complexity. All the visualization objects must offer are methods to increase or decrease their complexity.

Another policy only adjusts the visualization object while it is being modified by the user. During interaction, the object is represented with a less expensive (in terms of computing- and render time) representation. This can be obtained by lowering the level of detail (LOD) or simplifying the visualization algorithm itself [42]. When the interaction stops (i.e. the object moves out of focus) the object switches back to normal mode. This introduces a small delay, but it is usually small enough not to cause irritation. Faster computational algorithms are generally less accurate, implying a trade-off between accuracy and performance. The researcher must be aware of the fact that the simplified state of the object is possibly an incorrect representation of the data. He must decide whether the representation error is small enough to be useful.

2.3.4 Interaction

To control the visualization process, a graphical user interface presents the user with interaction methods. The visualization parameters can be changed, as well

as the physical properties of an object such as position, orientation and scale. In an immersive environment, interaction is often done by means of a 3D mouse, a wand or a data-glove. Its position and orientation are tracked and the researcher can invoke events by pressing buttons.

Standard navigation is performed by pointing the wand in the desired direction and pushing a button. Acceleration can be used, which increases the speed over time as long as the navigation button is pressed down. This enables faster transit between different areas, while delicate movements are still possible. Other buttons can be used to change the orientation.

Speech recognition allows a user to interact with an object without having to focus upon it. Recognizing human speech is not a straightforward task and computers have serious problems recognizing words if they are not pronounced in a certain specified way. The recognition error can be reduced by using only a subset of commands, depending on the context of the environment [43].

Chapter 3

FlowFish

This chapter describes the various visualization methods of FlowFish. The goal of FlowFish is described in section 3.1, followed by an overview of the architecture (section 3.2) and common implementation methods (section 3.3). FlowFish is designed around 3 classes of functions: supporting classes (section 3.4), global visualization classes (section 3.5) and local visualization classes (section 3.6). Each class is described in terms of its purpose, properties, advantages, disadvantages and implementation. The obtained results are presented in section 3.7.

3.1 Goal of FlowFish

Complex high-dimensional flow data demands a well designed representation environment. Immersing the scientist in the environment, allowing 3 dimensions to be used during representation instead of 2, improves the representations considerably since advanced methods such as motion-parallax can be used to analyze the flow. The data is often time-dependent and created by simulations. Updates of the representation should usually be performed as soon as updated data is available. Time-dependent representation methods should be available to analyze the flow over time.

FlowFish is a collection of flowfield visualization classes and tools to control them, suitable for immersive virtual environments. The FlowFish application employs this environment to unambiguously display complex high-dimensional flow visualizations. Several types of classes are available, each with their own specific parameters, interaction and visualization methods. Each class provides quick and intuitive methods to examine flows. FlowFish can be considered as a wrapper on top of existing visualization libraries, extending and combining the functionality of existing visualization classes and virtual reality interaction methods for visualizing flowfields in immersive environments.

A FlowFish object embeds a complete visualization network; from data input to visualized object. Interaction methods are included in each module to change an object's representation and behaviour. Each object functions as a so-called black box: data goes in, a visualization comes out. The internal visualization network is hidden from the user, except for the interaction methods required to control an object's settings. Updated data is loaded and processed immediately

so the visual representation matches the new state of the flowfield as quickly as possible.

The scientist is presented with the front-end of each module, which consists of the visualized data representation, as well as an interface to control the visualization parameters. The objects present themselves in the same manner as they are embedded in the FlowFish library: as one object, containing all the tools, filters and GUI items it requires to visualize and control the data.

While working in an immersive virtual environment, interaction by means of a mouse or keyboard is not sufficient. FlowFish provides suitable, intuitive tools to interact with the environment and to control visual representations. Objects inside and outside the flow can be selected, dragged and dropped. The interaction behaves in such a way as to imitate the grabbing and releasing of real life objects as best as possible. The contact between the researcher and the flow representation is direct, seemingly without the physical limitations present in the real world.

3.2 Architecture

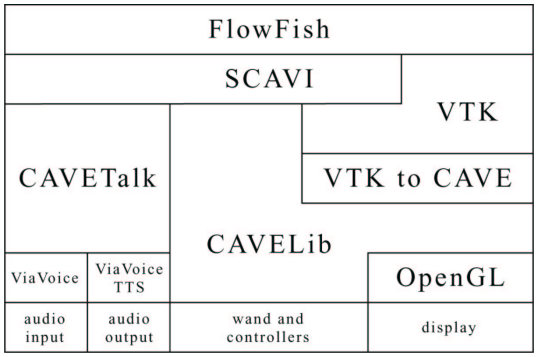


Figure 3.1: FlowFish architecture.

The visualization of data on a screen requires several layers converting the data to actual pixels (figure 3.1). The following sections shortly describe the functionality of each layer on which FlowFish is built.

OpenGL

OpenGL is an extensive set of portable graphic functions. It was introduced in 1992 and supports both 2D and 3D graphics programming [44]. Its application programming interface (API) is supported by a wide variety of computer platforms. Besides basic drawing routines such as rendering mathematical objects like lines and triangles, it supports texture mapping and special effects such as anti-aliasing. OpenGL-based applications can run on systems ranging from PCs and workstations to supercomputers. Many graphic boards support the OpenGL standard, providing fast rendering of large amounts of polygons in hardware which are necessary to create an immersive virtual reality environment.

CAVELib

The CAVELib library offers easy access to the VR system's hardware, including the interaction devices and stereo-display [45]. It provides access to the tracker data, transformation matrices and synchronization routines. A wide variety of display and interaction combinations are possible, allowing maximum freedom in the creation of a VR system.

VTK

The Visualization Toolkit is a free scientific visualization library supported on different platforms and operating systems [31]. Its functionality is improved and extended every day by many contributors all over the world. VTK includes over 500 C++ classes, including 2D and 3D visualization primitives, image processing, extensive mathematical functions, widgets, light sources and cameras. A visualization pipeline transforms numerical data into geometric constructs that are rendered into visual presentations. This concept is described in more detail in section 3.3.1.

VTK to CAVE

Matthew Hall has combined VTK with the CAVE library, thereby allowing the incorporation of VTK objects in a CAVE by means of a translation into OpenGL [46]. The polygonal data resulting from VTK objects is held in shared memory to make it accessible to all display processes. Dynamically changing data is handled automatically. VTK Rendering tools such as cameras and light sources are not supported and have to be performed in OpenGL directly. A number of changes have been made by the Section Computational Science of the UvA to the original distribution to support transparent and textured objects as well as to increase performance.

CAVETalk

ViaVoice is a speech recognition system which can interpret audio by means of a predefined set of commands. A user can speak through a wireless microphone and the CAVETalk library takes care of passing the interpreted commands from the speech recognition engine to the application. CAVETalk also offers audio feedback by means of the ViaVoice Text-To-Speech (TTS) synthesizer. When a command is carried out or finished, a sound or spoken text can be played through a set of speakers. CAVETalk is based on CAVESpeak, developed by Bram Stolk from SARA. CAVETalk supports dynamic loading of vocabularies based on application context to increase speech recognition performance [47].

SCAVI

The Speech CAVE and VTK Interaction library (SCAVI) created by Don Hannema, is a standardization of interaction tools in a virtual reality environment [43]. It contains a general purpose library for scientific visualization and object handling. Common visualization methods can quickly be integrated in an immersive environment by using the VTK-extended SCAVI-actor. It provides

basic positioning, rotation and scaling functionality by means of an input device such as a wand. Furthermore, SCAVI provides abstract input device event handling by processing all input devices and passing only desired events to the actors. A 2D menu class is available which can be used to access visualization parameters.

3.3 Visualization networks

A visualization network takes (scientific) data as input and provides a visual representation as output. The mapping of data values onto the domain of a representation parameter is an important part of the visualization process. The visualization pipeline (figure 3.2) is the basic implementation of the visualization network.

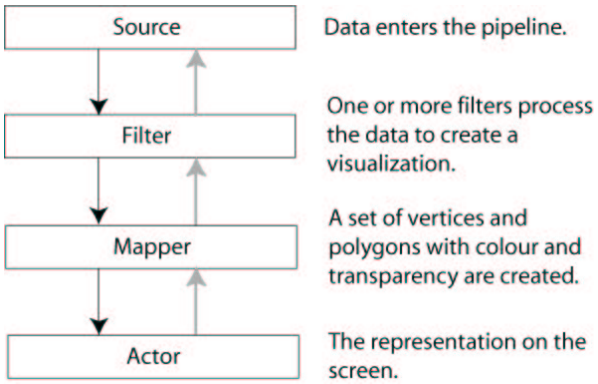


Figure 3.2: Visualization pipeline. Each square denotes a filter. The black arrows show the direction of data transfer through the pipeline. The gray arrows show the direction of update requests.

3.3.1 The visualization pipeline

A visualization pipeline is a first-in, first-out (FIFO) construction with a number of filters through which data flows in order to create a visual representation. Figure 3.2 shows an abstract representation of a basic visualization pipeline. The black arrows denote the direction of data flow through the pipeline. The gray arrows denote the direction of an update request through the pipeline.

A visualization pipeline consists of the following type of modules:

- Source. The data enters the pipeline by means of a Source object. Common examples are file-readers and objects that automatically generate data.

- Filter. Filters manipulate and process data or extract information. They extract a scientific representation from the data according to their settings or perform basic data functions such as merging separate data blocks.

- Mapper. The mapper converts its input into a set of polygons with colour, shading and transparency information.
- Actor. The actor is the representation of the visualization on screen and contains parameters necessary to add the visualization to the environment. The actor itself has no knowledge about visualization algorithms or parameters. It is fed by the mapper with vertices and polygons which it renders when requested to do so.

The flow of data through the network should be as efficient as possible. The amount of data must be kept to a minimum to reduce bandwidth congestion.

```
Update ( ) {
   if (preceding_object)
      preceding_object->Update ();
   if ( (input == modified) || (parameters == modified) )
      ComputeOutput ();
   else
      return;
}
```

Figure 3.3: Pseudo-code of the update of a pipeline object.

Figure 3.3 shows a piece of pseudo-code which takes care of the update of an object. If an object in the pipeline is requested to update itself, it will first notify its preceding object (if any) to update itself. Therefore this check-and-update-request travels automatically back to the beginning of the pipeline (the gray arrows in figure 3.2). When the preceding objects return, which indicates their current output is up to date, the object will check whether its own parameters or input has been modified and update its state and output when necessary. All objects following a modified object will notice their input data has been changed and update their state subsequently. Finally, the actor can provide the rendering environment with a set of polygons representing the current state of the visualization. An actor only invokes an update request on its visualization pipeline when the renderer requests its data. This method ensures updates only occur when they are requested. When for example an actor is invisible, no update requests will be performed.

In a visualization network, filters can be used to merge several visualization pipelines. A filter can combine the output of other filters to create its own output. Reusing the output of a filter reduces computation time and allows the creation of complex visualizations in which several datasets are used.

During the creation of a visualization network, the parameters of every object must be set to an initial value. Default values which work well for most datasets are set when an object is created. The environment must refrain from focusing upon a phenomenon from the start. The researcher must be aware of the fact that the initial visualization is generated by using default parameters which might not be correct with respect to the current dataset.

To reduce communication overhead and memory usage, part of the visualization network can be "cut off". For example, a reader can be destroyed when the data is loaded and is known to be static. Since filters do not know by which

objects their output is used, they have a reference counter. This counter is increased every time another object is connected and starts using its output. When the object is disconnected from a pipeline, the counter is decreased by one. As soon as it reaches zero, indicating no more pipelines are connected to it, it is destroyed and removed from memory.

The construction of the actual screen image from the end data of all visualization networks is performed by the graphics renderer, in our case OpenGL. It renders the pixels provided by the visualization networks to the screen taking into account the mutual spatial relations and visualization properties (colour, texture, transparency) of the visual objects.

3.4 Support classes

The support classes do not perform scientific visualizations but present modules that facilitate the creation of and interaction with visual representations.

3.4.1 A 3D menu: the `vtkCAVEMenu` class

To take full advantage of the 3 dimensional immersive nature of the environment, `vtkCAVEMenu` offers a 3D menu. The items consist either of extruded characters or polygon objects. Every item can have its own event handler which is invoked as the item is selected ("hitting and clicking" with the wand). Any polygon object can be used as a button. The size of these objects might differ greatly with respect to each other and the menu itself. To keep control over the overall menu-size, buttons can automatically be normalized. They are positioned in a grid-like or cubic structure with variable resolution and size.

Figure 3.4 shows a simple example of a `vtkCAVEMenu`, used by `vtkSourceCursor` (see section 3.6). The four spheres set the size of the cursor spheres, the letters D(ot), K(ube), P(lane) and C(loud) are used to set the shape of the cursor and the arrows ("<" and ">") respectively decrease and increase the cursor's resolution. The 8 diamond-shaped objects at the corners of the menu can be used to reposition and orient the menu.

3.4.2 General flow data access: the `cFlowField` class

The `cFlowField` class performs several tasks:

1. `cFlowField` depicts the flowfield by creating a bounding box around the flow data. This representation clarifies the spatial relationship between visualizations in the same flow. It facilitates the orientation of the user with respect to the flow and guides him in positioning items in the flow.

2. `cFlowField` controls initialization of the flowfield, such as spacing of the data; setting the size between subsequent data elements.

3. It takes care of loading data and merging separate datafiles when necessary. A uniform data structure is presented to the visualization classes so they need not to bother with reading, merging or reloading the data. The data describing a flowfield can reside in various formats. Multiple flow properties can be combined in one datablock. When a flow is simulated, separation of flow properties over several files prevents reloading

Figure 3.4: Perspective view of a `vtkSourceCursor` menu (see section 3.6).

those parts of the data which do not change, such as the geometry. Simulations can run locally or remote on dedicated computer-clusters. After the completion of a time step, the data is sent over the network to the visualization workstation [40]. The `cFlowField` class checks whether the input data is updated by means of a flag or timestamp. It will reload the data immediately and set a flag so subsequent filters are notified they have to recompute their data. These changes travel automatically from input-data through the visualization network.

4. Since the amount of data can be immense, subsampling is sometimes necessary. Out of every n voxels, only 1 is passed on[1]. This allows large datasets to be visualized without putting too much strain on the computing power or memory capacity of the computer. Visualization classes have access to both the original and subsampled data. This is mainly meant to be used by local visualization methods when extremely large datasets are used.

5. A visualization class might only use the vector-magnitudes of a dataset instead of the vectors themselves. For easy access to this scalar data-property `cFlowField` creates a volume with scalar data containing the vector magnitudes.

The mutual spatial relationship between visualizations must be kept intact so no distorted view is presented. Visualization methods which partially visualize the flowfield can be confusing when they are not properly aligned within the flow. To ensure that all visualization objects are correctly aligned with respect to each other, a global transformation matrix (TM) is necessary on which all

[1]Sampling discretizes a (continuous) signal. The *sample rate* indicates the number of samples taken from the domain per specified time period. Increasing the sample rate results in a higher resolution of the obtained discrete data.

Subsampling a dataset by a *subsample ratio* n means that of every n voxels of the original dataset only 1 is present in the subsampled data. This is done to obtain a smaller (in terms of memory usage) representation. Increasing the subsample ratio results in a lower resolution of the obtained discrete data.

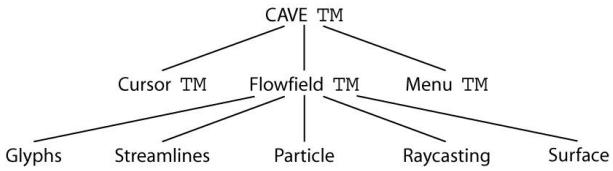


Figure 3.5: Scene graph of the FlowFish environment.

visual representations depend. A scene graph is used to describe the relationship between objects' transformation matrices built on top of each other (see figure 3.5). Changing the TM of an object located high in the graph will affect all the objects connected to it. The CAVE transformation matrix describes the transformations of the scenery to the viewer's eyes. The `vtkSourceCursor`, `cFlowField` and `vtkCAVEMenu3D` can be positioned independently of each other. All visualization classes use `cFlowField`'s matrix. Thus, when the flowfield is moved, all visualizations stay aligned.

In order to have several views of the same flow next to each other, several `cFlowField` instances can be created, using the same input data or the output of another `cFlowField` instance. Visualization methods belonging to the same `cFlowField` stay aligned while the flowfields can be repositioned, rotated or scaled with respect to each other. By manually creating each `cFlowField` instance, the user is aware of the fact that more than one collection of representation methods is used, which all visualize the same flow but do not have any spatial relationship.

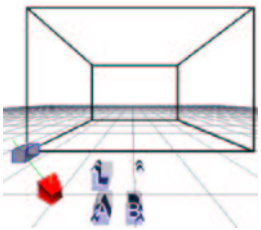


Figure 3.6: Flowfield representation with anchor and 3D menu.

The only visualization `cFlowField` performs is the creation of a bounding box around the data, possibly in combination with an anchor (figure 3.6). The bounding box or anchor can be used to move, rotate and scale the flowfield representation. They directly affect the main transformation matrix, so that all visualization objects stay aligned.

Interaction

Through the `cFlowField` menu, the interaction with the flow can be changed:

1. The bounding box can be used to move and orient the flowfield by means of the transformation matrix. It provides easy access since the flow is usually the largest object in the virtual environment. However, objects inside the flow can not be accessed.

2. The anchor allows both access to the flow's transformation matrix as well as to objects positioned within the flow area.

3. The flowfield can be locked which prevents access to its transformation matrix by means of interaction. All other objects remain accessible.

Implementation

The data is read by `vtkDataSetReader`s or passed directly on by means of shared memory segments (figure 3.7). Gray boxes denote the objects with direct communication to other objects or to the user by means of CAVE or SCAVI methods. Data can be divided in a geometry, vector and scalar domains. This allows partial update of the flow, reducing network overload and computation time. When they are combined in one file, they are read in by the geometry dataset reader and both v(ector) and p(ressure) reader objects are ignored. To create a subdomain of the data using a subsample ratio, `vtkExtractVOI` filters remove part of the data which is merged again by the `vtkMergeFilter` to combine geometry with vectors and scalars. `vtkOutlineSource` creates a bounding box for the flow. `vtkCubeSource` is either set to a small cube-shape in the corner of the flow's bounding box to serve as anchor or aligned with the bounding coordinates of the geometry file. Through the `vtkCubeSource` the user can adjust the main transformation matrix in order to reposition, orient or scale the flow.

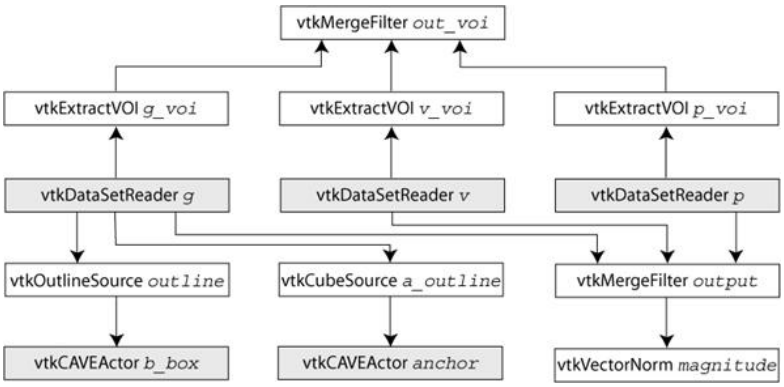


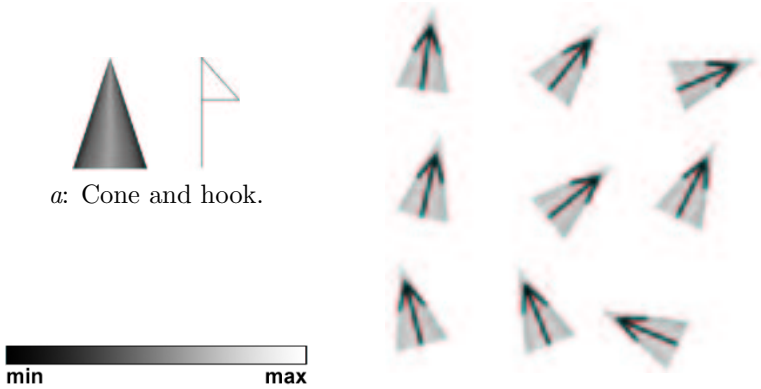
Figure 3.7: `cFlowField`'s visualization network.

3.5 Global visualization methods

The tools which are used to examine a flowfield can roughly be divided into two separate visualization methods: global and local methods. Global methods provide an overall view on the behaviour of the flow, local methods give a more detailed representation of certain locations within the flow. These locations can be interactively set by the user.

3.5.1 Glyphs: the `cGlyphsFF` class

Particle methods create one or more polygon-based objects which visualize both scalar and vector properties of the flow. The physical appearance (orientation, size, colour) of these objects is changed according to the value of the flow-properties at the corresponding position.



a: Cone and hook.

b: Gray lookup table.

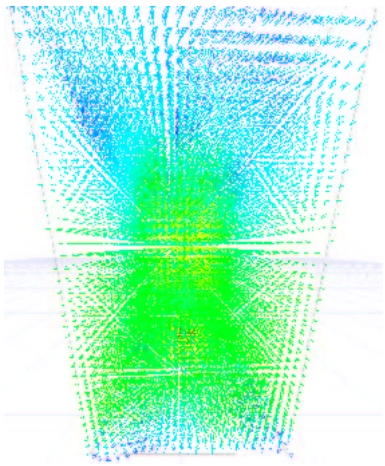
c: Glyphs (cones) oriented to vectors.

Figure 3.8: Glyph objects, standard lookup table and orientation example.

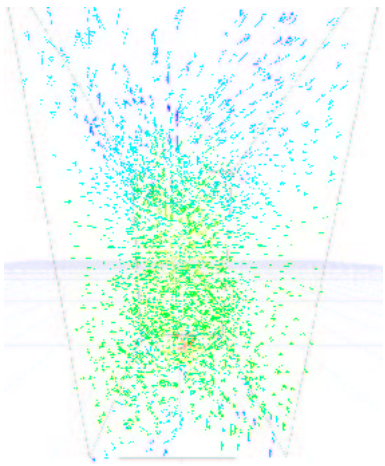
Glyphs are hook- or cone-shaped particles (figure 3.8-a), pointing in the direction of the flow by aligning themselves with the direction vector at their position (figure 3.8-c). They are useful in providing a global overview of the flow's behaviour. Additional flow characteristics can be visualized by colouring or scaling of glyphs according to scalar flow data such as pressure or speed. A lookup table is used to map voxel values to hue-, saturation- and opacity values when colouring of the glyphs is performed (see 3.8-b). For example, a mapping from "low" blue values to "high" red values is common. Unless specified otherwise, all screenshots use this configuration.

Scaling is useful for highlighting minimum and maximum values of a flow property. Relatively large glyphs or seemingly void areas positioned at unexpected positions help to locate the presence of such values. Extremely small or large vectors can be present, for example caused by the pulsatile nature of the flow. While scaling, this can result in a few extremely large glyphs. Therefore the glyphs are normalized but this might render a lot of particles almost invisible. Glyph-objects can therefore be clipped to a specified size so their size cannot exceed the clipping value. This allows smaller glyphs to be increased in size while the largest glyphs remain the same. Clipping must be used with extreme caution since it can cause a non-linear mapping from flow data values to glyph size thereby deforming the correct visual representation.

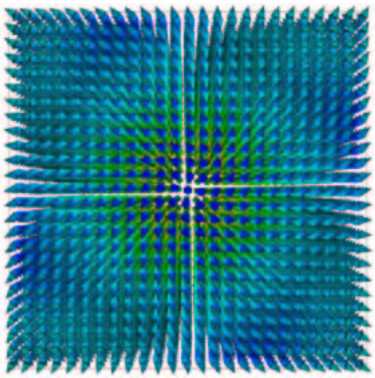
Flowfields often consist of millions of data points. Creating a glyph at every point will clutter the view immediately (figure 3.9-a and 3.9-c). A global subsampling of the data, provided by `cFlowField` can be used, but this will not always be sufficient: its output consists of a regularly spaced dataset used by many visualization filters. Glyphs generally require higher subsampling rates than other visualizations. Figures 3.9-b and 3.9-d show subsampled representations. When seen in an immersive environment, these visualizations provide



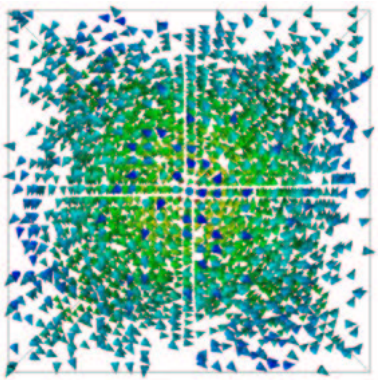
a: Original dataset (hooks, front).



b: Subsample ratio of 8.



c: Original dataset (cones, bottom).

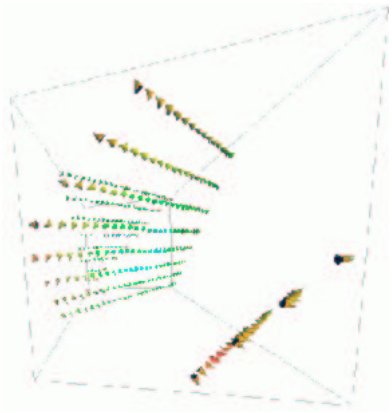


c: Subsample ratio of 7.

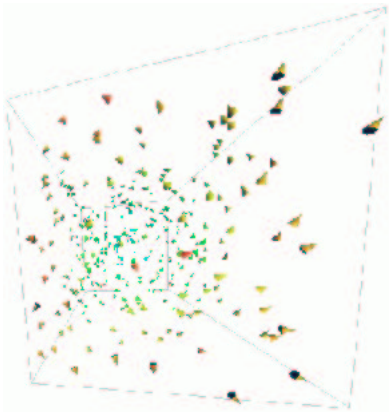
Figure 3.9: The effect of subsampling on glyph visualization of a whirlwind simulation (see page 72 for colour versions).

a global view of the rotating motion of the dataset, without cluttering the view. With certain sub-sampling values aliasing of the data might appear (figure 3.10-a). Random sub-sampling avoids this: instead of taking every n^{th} sample, a random sample from every set of n voxels is taken. The average distance between sub-sampled points remains the same, while reducing the chance of overlooking small phenomena.

If the size of an interesting flow phenomenon is smaller than the distance between sub-sampled points, it might not be revealed in the simulation. Random sub-sampling reduces this risk, but the scientist should always be aware of the possibility of overlooking an interesting phenomenon. When time-dependent flow is visualized and a subsequent snapshot is loaded, the random positions of glyphs are recomputed, so over time all voxels within the dataset are visualized an equal amount of times. When the dataset is often updated, this might result in "jumping" glyphs which often change of position. This can create a confusing



a: Aliasing effect in a regularly subsampled dataset.



b: Random subsampled dataset, preventing aliasing.

Figure 3.10: Aliasing effect during subsampling (top-view) (see page 73 for colour versions).

view, therefore random subsampling can be turned off.

The total amount of polygons and vertices of the glyph objects can be quite high, which can slow down the rendering speed considerably. Sub-sampling is only possible to a certain degree; too much information will be lost if a relatively high sub-sample value is used. To lower render times, a simplified glyph object can be used, such as the hook shown in figure 3.8-b. The hook consists of only 4 vertices and no polygons, leaving the triangle transparent. The cone consists of 11 vertices and 12 shaded polygons. By shading the polygon, cones provide a good depth-cue on their orientation, but are costly in terms of render time. The orientation of the hooks is less obvious as it is not shaded, but the amount of vertices is considerably lower, allowing them to be rendered faster.

Interaction

Using `cGlyphsFF`'s `vtkCAVEMenu` the following options can be selected:

- Select hook glyph or cone glyph.
- Turn sampling on / off.
- Set subsample ratio by means of a keypad-like menu.
- Turn random sampling on / off.
- Colour by scalar or vector magnitude.
- Turn constant scaling on / off.
- Scale by scalar or vector magnitude.
- Increase / decrease constant scaling. With two buttons the constant scaling factor can be doubled or halved.

Implementation

The `cGlyphsFF` class is derived from VTK's `vtkGlyph3D` class. The `vtkStructuredPoints` object contains the output of `cFlowField`. Again, all gray boxes denote the objects with direct communication to other objects or to the user by means of CAVE or SCAVI methods. The `SetInput` method of `vtkGlyph3D` is bypassed to route the data through a `vtkMaskInput` filter which subsamples it. This filter only passes points and point-data through, so in order not to loose any scalar information, the original data is probed with the subsampled points. If no subsampling is used, the input data is passed directly to the `vtkGlyph3D` object. It creates the actual glyphs using either the 3D cone primitive or 2D hook. Polygon normals, created by `vtkPolyDataNormals`, are required to create smoothly shaded polygons in the final representation. The colouring of glyphs is performed by mapping scalar data onto a lookup table. The mapper, which is included in the `vtkCAVEActor`, uses this map to colour the final polygon data. The `vtkCAVEActor` is the representation of this data in the VR environment and is responsible for the position, orientation and scale of the glyphs.

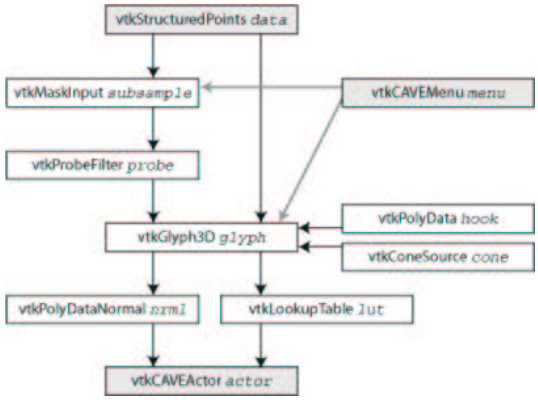


Figure 3.11: The glyphs class visualization network.

3.5.2 Raycasting

Volume rendering methods use scalar data of a flowfield to create a volumetric representation. Two methods of volumetric scalar rendering are discerned: raycasting and surface rendering.

Raycasting includes every voxel in its representation of the flow. No subsampling is performed, so the risk of overlooking interesting phenomena is reduced. It visualizes volumes and surfaces consisting of scalar values which are of interest to the user. The partial transparency of the raycasted image enables an overview of the flow and facilitates a comparison between spatially scattered regions with equal scalar values.

The raycasting technique is based upon the idea of "shooting" a ray from the position of every pixel in the final render-frame towards the data. Every voxel receives a property-value (such as transparency or colour) based upon a lookup table. Every voxel through which the ray travels, adds its value. The

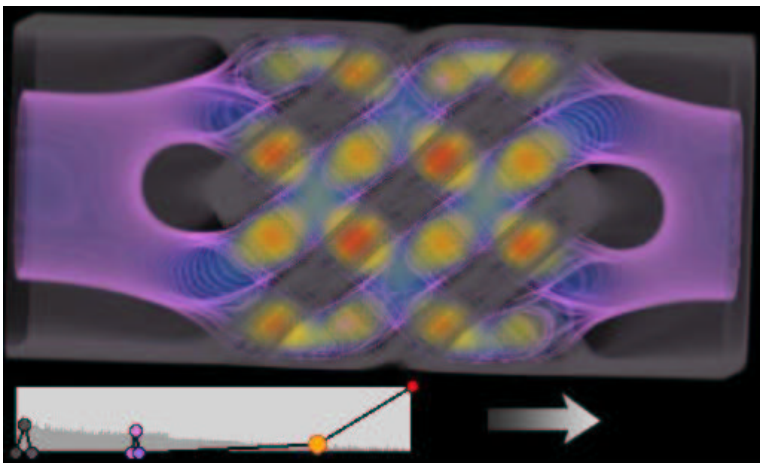


Figure 3.12: Raycasted image of a 3D static reactor (SMRX) filter (see section 3.7 for a description) [48]. The arrow denotes the flow direction (see page 75 for a colour version).

ray is terminated when it leaves the data-domain and the pixel from which it originated, receives its final value. To get an effective and accurate image, the user must find a definition of the lookup table, such that features of interest are highlighted in the resulting image. Every flow has its own behaviour and every analysis has its own goal, therefore a useful parameterization of a lookup table must be defined separately for each flow. Furthermore, the scalar range of different snapshots of a time-dependent flow can differ considerably. The user might need to correct the lookup table to adjust it for the new scalar range.

Opposed to the particle methods, the results of raycasting can only be cached as long as the observer's location and orientation do not change with respect to the dataset. As soon as movement of either object occurs, the casted rays will travel at a different angle through the dataset and will therefore have to be recomputed.

Raycasting is computationally expensive and demanding. There are several methods to speed up the raycasting technique and discard unnecessary rays [49]. Early ray termination is a technique that can be used if the rays are traversed front-to-back. It ends the ray traversal after the accumulated transparency or colour for that ray is above a certain threshold. Octree decomposition is a hierarchical spatial enumeration technique that permits fast traversal of empty space, thus saving substantial time in traversing the volume. Adaptive sampling tries to minimize work by taking advantage of the homogeneous parts of the volume; for each square in the image, one traverses the rays going out of the vertices of the bounding box and recursively goes down repartitioning this square into smaller ones if the difference in the image pixel value is larger than a certain threshold.

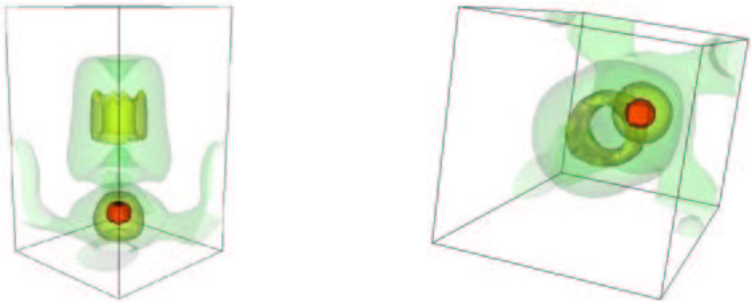
Dedicated workstations such as a Silicon Graphics Workstation [50] have hardware support for raycasting, but of-the-shelf graphics cards, such as used in the DRIVE system (see section 2.3.1) do not always support this. Performing raycasted images solely in software is computationally very expensive and

will effectively render smooth interaction impossible. Since the DRIVE system lacks software support for raycasting methods in virtual environments, this visualization method was not implemented in the FlowFish library.

All raycasted images of this document were created by Kitware's VolView application [51]. VolView is a 2D application which creates high resolution raycasting images of VTK data on common workstations.

3.5.3 Iso-surface rendering: the `cSurfaceFF` class

Surface rendering visualizes a threshold or boundary of scalar flow data by producing a so called iso-surface within the dataset [52]. Iso-surfaces provide an intuitive visualization when the user wants to find the location of a specific scalar value or boundary within the flow. Iso is short for iso-parametric: The iso-surface connects all the voxels with the specified iso-value (see figure 3.13). Between adjacent voxels with values above and below the iso-value, linear interpolation is performed.



a: Low, mean and high value iso-surface *b*: Top-perspective view.

Figure 3.13: Iso-surface visualization of the vector-magnitudes (speed) of a whirlwind simulation (see page 73 for an enlarged version).

Visualizing several surfaces with different iso-values requires the surfaces to be at least partially transparent since they could enclose each other. The transparency and colour of a surface can be fixed to a specified value, depending on its iso-value, or can be computed by mapping another scalar property of the data onto a transparency and colour lookup table.

Iso-surfaces can be approached by raycasting: using a so-called spike in the lookup table, narrowing the opacity-range around the desired value, only voxels with the desired iso-value can be highlighted. Raytracing does not perform any interpolation between adjacent voxels though, therefore holes are likely to appear in the surface. Furthermore it is costly in terms of computing time and power.

To create the polygonal surface, the marching cube algorithm is usually considered to be accurate, though not extremely fast. 8 voxels positioned in a cube are considered. Their values are examined to check whether the surface crosses the cube, which it does if some of the voxels have values below the specified iso-value and some values are above. The cube is then triangulated. Using symmetry, 14 unique cases are possible (figure 3.14). The disadvantage is one case where the surface cannot be determined by looking only at the voxels'

values (see figure 3.15). Using the surface normals of surrounding triangles helps to overcome this problem.

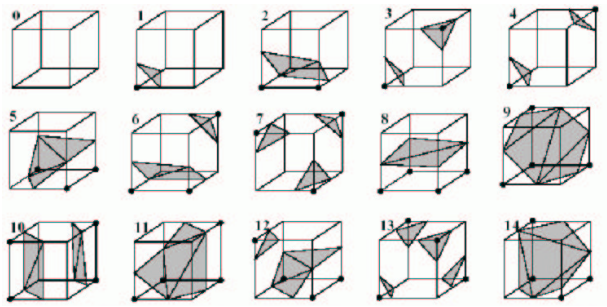


Figure 3.14: The 14 unique triangulations of the Marching Cube algorithm (no triangulation is performed on the empty cube #0).

a: 2D situation. *b*: 3D situation.

Figure 3.15: Ambiguous triangulation.

As long as the flowfield does not change, the polygons describing the surface are stored in memory. A surface visualizes a single time step of time-dependent flow. When a subsequent snapshot is loaded, the surface must be recomputed. This might create a confusing representation when surfaces with the same iso-value do not seem to have any spatial relationship across several snapshots. For example, a shockwave caused by pressure will not have the exact same iso-value, so visualizing it by an iso-surface will probably not show a consistent surface moving along with the shockwave over time.

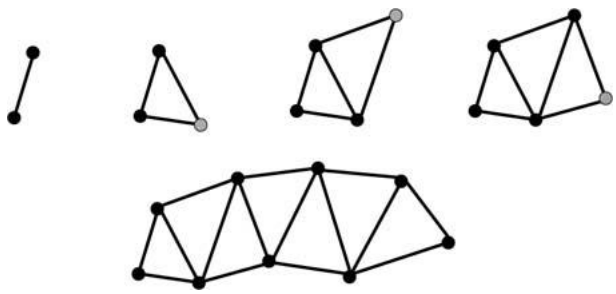


Figure 3.16: The creation of triangle-strips.

Depending on the size of the flow and the choice of iso-value(s) the resulting amount of polygons is considerably high. Several algorithms exist that reduce the number of polygons while keeping the mesh intact as much as possible. Decimation reduces the amount of triangles by merging polygons connected by an angle less than a specified threshold [52]. Triangle-strips reduce the amount of

memory by storing less information by combining vertices of adjacent triangles (see figure 3.16). First, triangulation breaks down polygons into numerous triangles. Normally, each triangle consists of 3 vertices, so $3n$ vertices are necessary to store n triangles. Triangle-strips start with 2 vertices and add 1 vertex per adjacent triangle, resulting in a total amount of $n + 2$ vertices for n triangles.

Implementation

The visualization network for the `cSurfaceFF` class is shown in figure 3.17. The `vtkContourFilter` at the top creates the surface from the dataset using the Marching Cube algorithm. The `vtkDecimate`, `vtkTriangleFilter` and `vtkStripper` filters reduce the amount of polygons and data to enable smooth interaction. `vtkProbeFilter` probes the resulting polygons with the original dataset for scalar data. The iso-surface can be coloured by mapping this flow-property on the `vtkLookupTable`. The `vtkPolyDataNormals` normalizes vectors so lighting is correctly applied. The user specifies the iso-value(s) and controls various parameters of the decimating filter, implicitly controlling the granularity (increasing the polygon reduction (rough surface) or decreasing (smooth surface)) of the iso-surface appearance which is inversely proportional to the amount of frames per second when the surface is the main visualization method.

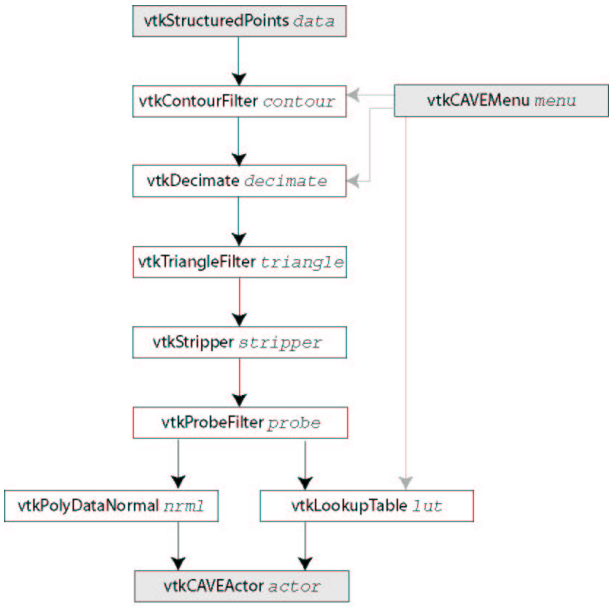


Figure 3.17: Visualization network of the `cSurfaceFF` class.

3.6 Local visualization methods

Local visualization methods visualize a part of flow data. A cursor object is present to set the location from which the visual representation is created.

3.6.1 3D cursor: the `vtkSourceCursor` class

Several objects require a 3D cursor to select a specific location or locations in the flowfield. The `vtkSourceCursor` class offers several cursor-types, such as a single dot, a plane, cube or cloud of spheres (figure 3.18). Type, size and resolution can be set by means of a `vtkCAVEMenu` menu (see figure 3.4). The cursor is positioned and oriented by dragging one of its source-spheres to the desired location. The location of each sphere in the cursor specifies the location of a point-source that is used in the local visualization methods.

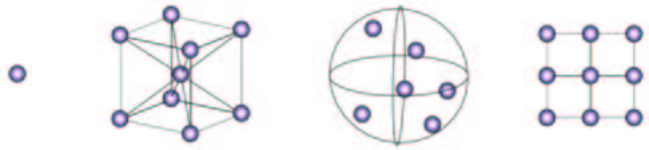


Figure 3.18: `vtkSourceCursor:` The four shapes of the cursor: dot, cube, cloud (resolution 6) and plane (resolution 3).

The flow has its own coordinate system which is initially aligned to the CAVE coordinates but can be altered by the user by dragging the bounding box or anchor. The scene graph of figure 3.5 shows the independent positions of the flowfield and the cursor. The coordinates of the `vtkSourceCursor` spheres must be translated back to CAVE coordinates by inverting its own matrix, and from there to the flow's coordinate system. This is done automatically by each visualization class which provides its transformation matrix to `vtkSourceCursor`.

Interaction

`vtkSourceCursor` provides a menu through which the following options can be selected:

1. Select the shape: dot, cube, cloud or plane.
2. Set the size of the cursor spheres to one of 4 predefined sizes.
3. Decrease / increase resolution (only effective while a dot or plane shape is used).

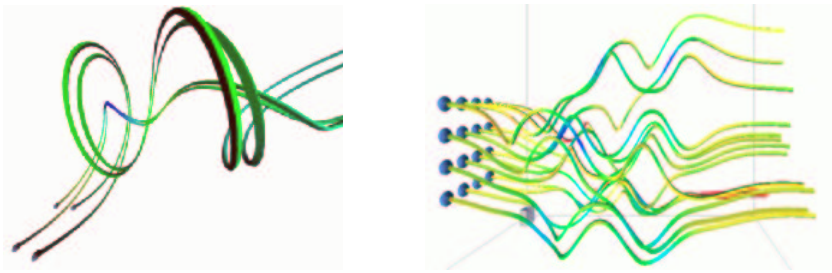
The size of the cursor can be adjusted in the same way other visual objects are adjusted: by clicking with the middle wand button, a red bounding box appears around the cursor. By selecting and dragging with the left wand button, the size is increased or decreased.

3.6.2 Streamlines: the `cStreamlinesFF` class

Glyphs are the only method of the previously described visualization methods that provide information about the flow direction. They fail to clearly visualize the behaviour and spatial dependence between specific locations within the flow. Both streamlines and animated particles aid the user by visualizing curving motion and converging and diverging flow behaviour. A streamline wrapping itself around a structure or sliding through obstacles provides a very intuitive

understanding of the flow. By using several streamlines, the researcher can investigate how the flow curves and whether it diverges or converges at specific points.

Streamlines represent the path of a massless particle through a vector field *in a single snapshot of the flow data*. The streamline does not visualize the path of a particle over time. A particle follows the motion of the flow within a single snapshot, originating from one or more points set by the user by means of a `vtkSourceCursor`. As the particle propagates, it leaves a trace, called the streamline. They are represented by tubes placed inside the flow data (figure 3.19).



Whirlwind. Mixer.

Figure 3.19: Streamline examples (see page 74 for colour versions).

A tube with rectangular shape is wrapped around the path. Its shaded surface helps during analysis of the streamline's shape. By rotating it around the eigenvectors of the streamline the vorticity ("twirling" motion) can be represented. The tube can be coloured by a scalar, for example to denote speed or pressure. In addition, its radius can be varied by scalar data.

A streamline is generated by placing a virtual massless particle in the flow. The particle is released and integrates through the flow from its initial position. A specified step length creates vertices on the particle's path. Several integration schemes are available. Common methods are Euler's method and the n^{th} order Runge-Kutta method:

- Euler's method consists of using only the first term of the Taylor Series Expansion:

$$y_{n+1} = y_n + f(x_n, y_n)\Delta x \tag{3.1}$$

The so-called truncation error in Euler's method arises because the curve $y(x)$ is usually not generally a straight line between neighbouring points x_n and x_{n+1}, which is assumed in equation 3.1. The main reason of the large errors in Euler's method is that the method only evaluates derivatives at the beginning of the interval, i.e. at x_n. The truncation error can be assessed by expanding $y(x)$ about $x = x_n$ while using a step size of h [53]. Integrating over a finite interval creates a net truncation error of $o(h)$ indicating the error is directly proportional to the chosen steplength.

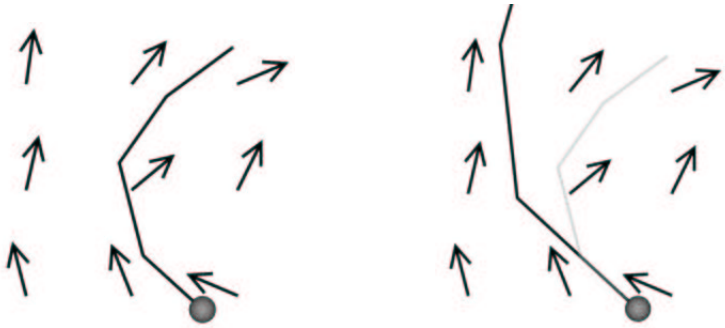
- Presented here is the 2^{nd} order Runge-Kutta method [54]. This method is not such an asymmetric integration scheme as Euler's method because it makes an Euler-like integration step to the middle of the interval. Take h as the chosen interval between x_n and x_{n+1}, then:

$$k_1 = hf(x_n, y_n) \tag{3.2}$$

$$k_2 = hf(x_n + h, y_n + k_1/2) \tag{3.3}$$

$$y_{n+1} = y_n + k_2 + O(h^3) \tag{3.4}$$

As indicated in the error term $o(h^3)$, this symmetrization cancels out the first-order error. The error-reduction can be improved by taking more than one integration step in the interval. The 4^{th} order Runge-Kutta integration scheme is one of the most commonly used integration methods. The integration of the streamlines in the `vtkStreamer` class through the dataset is by default performed by a 2^{nd} order Runge-Kutta method but can be set to a 4^{th} order. Both methods give accurate results while keeping the computation time relatively small.



Small integration step. Large integration step.

Figure 3.20: Streamline integration with different integration steps.

The integration step combined with the flow speed determines the detail of the particle trace. A small step creates a more detailed streamline but takes longer to compute. A higher integration step might overshoot interesting areas when a sharp curve is present in the flow data (see figure 3.20). This can result in streamlines which do not seem to follow the flow at all (figure 3.21). There is no clear solution for parameterizing a streamline. Streamlines are computed via numerical integration, they are therefore only approximations to the "real" streamlines. The accuracy of the streamline depends directly on the integration step. Decreasing the integration step will increase the accuracy. In order to allow a smooth interaction with the environment, the involved computations must be finished fairly fast. Technical restrictions such as necessary computation time and render time of the resulting polygons in combination with the flow data pose limits on the minimum integration step. An integration step which is set too low might create so many points on the streamline that it slows down the computation and visualization beyond effective performance. Careful manual adjustments by the user, possibly provided by hints of the environment

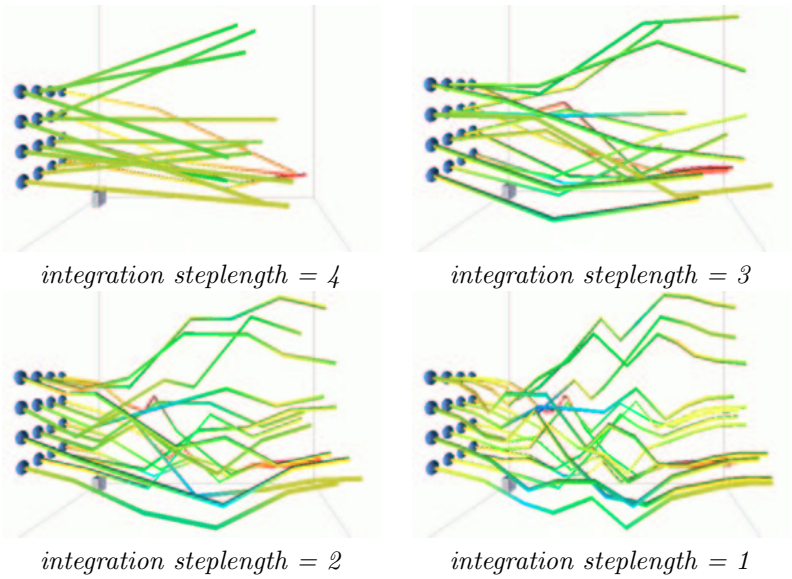


Figure 3.21: The effect of the integration steplength on streamline generation.

concerning minimum, maximum and mean vector-magnitudes present in the data, will always remain necessary to find a good compromise for the integration step.

The streamline must satisfy three criteria in order to compute and add a subsequent vertex:

1. The speed, the magnitude of the direction vector, must be higher than a certain user-specified threshold, called "terminal velocity".

2. The last computed vertex must lie within the data boundaries.

3. The number of integrations, expressed in the number of vertices, must not exceed a user-specified limit.

These 3 criteria prevent the creation of an enormous amount of vertices on a very small scale which increases computation and render time considerably without adding accurate information to the visual representation. The terminal velocity can be set automatically by computing the minimum and maximum vector-magnitudes and expressing the terminal velocity as a percentage of these magnitudes. It can be assumed that the minimal velocity magnitude, when compared with the maximum vector magnitude, usually approaches zero. Therefore the terminal velocity can be expressed as a percentage of the maximum vector magnitude present in the flow data.

Care must be taken when time-dependent flows are used. Maximum vector magnitudes can differ considerably between subsequent snapshots of the flow, for example during the startup-phase of a simulation. When the terminal velocity is computed automatically, this can result in a confusing representation, such as streamlines which are terminated quickly, while in subsequent snapshots they extend much further.

Interaction

To speed up the rendering process, the streamline can be visualized as a wire-frame line (wire-line), instead of a tube. The tubes have the advantage of a shiny surface by which the banking and twirling motions of the streamline are immediately obvious, but they increase render time considerably. To take advantage of both rendering methods, the user can choose to turn off tube-rendering while he is repositioning the `vtkSourceCursor`, and turn it back on when the cursor loses focus. This enables a smooth interaction, while not loosing the benefits of tube-visualization. Furthermore, a temporary higher step size might be used. This does not affect the computation of the actual streamline, but the distance between points which are used to visualize the stream. A higher step size creates a "rougher" appearance of the streamline, reducing the amount of resulting polygons and vertices. When the user keeps in mind that the temporary visual representation might show "short-cuts" not actually present in the real streamline, this will not result in a confusing image.

Through the menu, the following options can be selected:

1. Turn "always wire-line" on/off. When set, this option will always show the streamlines as wire-lines, even when they are not selected. By default, wireframe-lines are used when the cursor is being dragged, otherwise tubes are visualized.

2. Turn "never wire-line" on/off. When set, this option will always render tubes.

3. Increase / decrease integration steplength by 50%. By adjusting the integration steplength relatively to its previous value instead of setting the absolute value, the user does not need to know exact flow data values, such as minimum and maximum vector magnitude.

4. Increase / decrease time step by 50%.

5. Turn speed scalar calculation on/off.

6. Turn varying of the radius by scalar value on/off.

7. Turn vorticity calculation on/off.

Implementation

The `cStreamlinesFF` class is derived from VTK's `vtkStreamLine` class (figure 3.22). Its input consists of the flow data and the starting positions provided by the `vtkSourceCursor` object. The `vtkTubeFilter` generates a smoothly shaded tube object around the streamline. When vorticity scalars are generated by `vtkStreamLine` the tube visualizes the vorticity by radial rotation around the streamline. The `vtkPolyDataNormals` provides the polygon normals to apply correct lighting to the surface. When interacting, the creation of tubes is turned off and the generation of the streamlines is provided directly as input to the mapper of the `vtkCAVEActor` to increase the number of framerates per second during interaction.

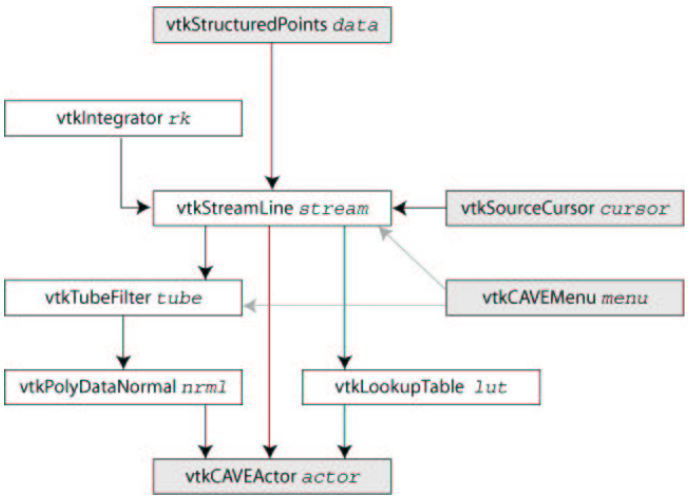
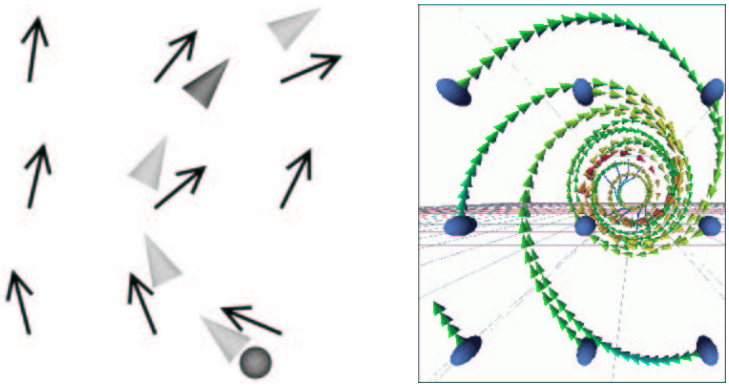


Figure 3.22: The streamlines class visualization network.

3.6.3 Animated particles: the `cParticlesFF` class

Particle animation is a dynamic version of the streamlines visualization. A glyph-like object -the particle- is animated along the flow's direction using the flow speed (figure 3.23). This visualization method shows one property of the flow which streamlines are less capable of visualizing; speed. A particle slowing down or speeding up through an area provides a far more intuitive impression of flow speed compared to, for example, colouring of streamlines.



a: Particle propagating through a dataset.

b: Bottom-up view of a whirlwind simulation.

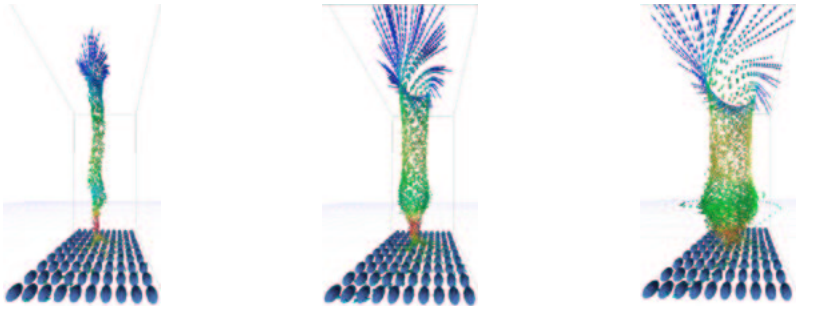
Figure 3.23: Animated particle propagation and static particle example (see page 74 for an enlarged version).

Particles can be either static or dynamic. Static particles are fixed: they are separated by a specified time step. Increased vector-magnitudes result in a larger distance between subsequent particles. Animated particles "fly" through the flow. One animation step propagates the particles once using the current

vectors of the flow. Increased vector-magnitudes will make them move faster. When a particle reaches the end of its streampath, it is reset to the cursor sphere from which it originated. When using several particles simultaneously it can be difficult to get a clear view of the flow-direction in a single snapshot, but by using particle animation in combination with streamlines, this problem can be overcome.

Another advantage of animated particles is that they visualize the streampath of a particle over time. When animated particles are used to visualize a time-dependent flow, they automatically use new data as soon as it is available. A `cFlowField` object takes care of reloading the data when the next time step has arrived. When the particles' actor is requested to update itself, the update request travels back to the input dataset provided by `cFlowField`. Since this data is modified, the visualization pipeline of the particle will update itself using the new data. An important distinction must be made between static and animated particles when such a time-dependent flow is represented. Like streamlines, *static* particles compute their path *within a single time step. Animated* particles compute their path using present data. The path will be the same as a static particle's path when no subsequent snapshot of the flow is loaded between particle updates. But a massless particle's path over time will be visualized when the flow data is updated particle animations, which is not likely to be exactly the same path a static particle follows in a single timestep.

Additional scalar flow data can be visualized by colouring and scaling the particles according to the value of the flow property or speed at their position. The range of the lookup table can be adjusted automatically. This may give unwanted results when only a few particles are visualized: the range of their minimum and maximum scalar values (either computed by `cParticlesFF` or already present in the flow data) can be a small subdomain of the global scalar domain of the flowfield: this results in particles that are scaled or coloured by a local minimum or maximum which differs considerably from the global minimum or maximum value.



integration step 0.8 *integration step 1.6* *integration step 3.2*

Figure 3.24: Impact of integration steplength (see page 74 for colour versions).

Both kind of particles use the same integration method as streamlines. While choosing the integration step, the same problems (as mentioned in the streamlines section) arise. Figure 3.24 shows the same flow data visualized with different integration steps.

When the dataset is large, a lot of integration points can be created during

the computation of the streampath, which might slow down the rendering and animation processes. The time step value can be used to skip one or more integration points between neighbouring particles, thereby reducing the total amount of particles. By increasing the time step value, a rougher visualization is created, but it uses the original particle-path. This does not affect the accuracy of the particle's path, which is generated with the integration steplength.

Interaction

Since particles do not use much time to render, they are not represented in a simplified manner while the cursor is being dragged. Options accessible through the menu are:

1. Double or halve the integration steplength.
2. Double or halve the time step between subsequent particles.
3. Turn the generation of speed scalars on or off.
4. Set colouring to colour by vector or scalar.
5. Set scaling to scale by vector or scalar.

Implementation

`cParticlesFF` is based on the `vtkStreamPoints` class which in its turn is derived from `vtkStreamer` (figure 3.25). The animated particles are generated by the `vtkStreamPoints` object. The interpolation of animated particles can be performed by different algorithms such as a linear method or a second order Runge-Kutta method as described in section 3.6.2. By default the 2^{nd} order Runge-Kutta method is used. `vtkStreamer` generates scalar data by computing the speed of the stream. The `vtkProbeFilter` allows access to scalar data already present in the original data. When it performs no probing, the speed scalars are passed through. `vtkGlyph3D` takes the particles' data and creates the polygon objects which represent the particles in the virtual environment. They can be coloured or scaled by scalar data present in the original data or as computed by `vtkStreamer`. `vtkPolyDataNormals` adds the normals to the polygons so particles will appear correctly shaded. The colouring is performed by mapping a scalar data-property onto `vtkLookupTable` which passes the appropriate colour to the mapper included in `vtkCAVEActor`.

3.7 Results

These sections describe the tests obtained with FlowFish's visualization methods and several datasets. Performance and qualitative measurements were taken. Performance was measured by logging the number of frames per second as provided by the CAVELib library. All tests were carried out on the UvA-DRIVE system (see section 2.3.1). The qualitative measurements of the visualizations are obtained by comparing the visualizations with the known content of the flow data. When a flow is known to contain certain phenomena and to show a certain behaviour, the visualization method must provide the scientist with a

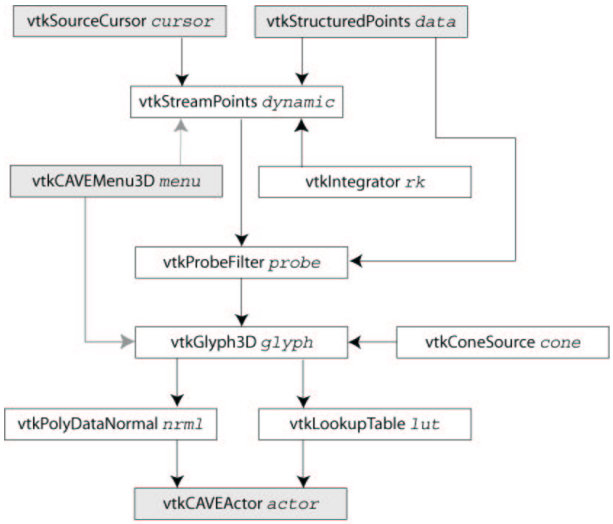
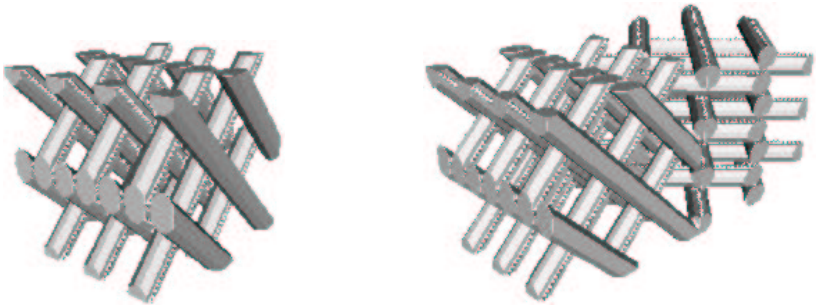


Figure 3.25: The animated particles class visualization network.



a: single mixer. *b*: double mixer.

Figure 3.26: Mixer geometry.

representation which allows him to analyze the flow, its behaviour and to locate and extract these phenomena.

The various flow data sets used in the experiments differ in size and originate from different scientific flow simulation areas. This way it is easier to obtain general results without focusing too much on a specific flow analysis or simulation problem. The datasets used during the tests are:

- von Karman Reactor. This 2D dataset represent the flow around a block-structure with a speed of $\tau = 0.9$ at which the flow oscillates and shows vorticity. Simulated using the Lattice-Boltzmann method by Arjen Schoneveld. Since this is a 2D dataset and therefore a bit trivial to represent concerning the visualization in an immersive virtual environment, no snapshots of this dataset are provided. It was included in the performance tests because of its small size, which hardly requires any subsampling. Number of voxels: 7,500.

- Whirlwind. This 3D dataset contains a simulated whirlwind. The flow

makes circular motions with different speed, depending on the distance from the center. The speed increases from the outside towards the middle of the whirlwind. The nexus of the whirlwind goes almost straight up, hardly rotating around its z-axis. The dataset contains no geometry, only flow. Number of voxels: 25,000.

- Stenosis. A 3D dataset simulating a stenosis in an artery (see section 4.1). A high pressure is to be expected just before the stenosis, while the highest velocity should be located in the most narrow area. The pressure and velocity distribution should be symmetric around the flow axis of the artery. The artery was generated by using a voxel modeling technique called "distance sampling". The fluid is simulated by using the Lattice-Boltzmann method. Number of voxels: 30,000.

- Single static mixer. This is a SMRX simulation created by D. Kandhai e.a. [48]. It contains a static mixer which is used to mix 2 fluids with an extremely high viscosity in a static reactor (see figure 3.26-a). The fluids enter the mixer with a high velocity; the fluid which exits the mixer should contain a mix of those two fluids. The fluid is simulated by using the Lattice-Boltzmann method. Number of voxels: 468,968.

- Double static mixer. 2 single static mixers positioned orthogonal, using the same simulation environment as the single mixer (figure 3.26-b). Number of voxels: 699,608.

3.7.1 Results of support classes

General flow data access

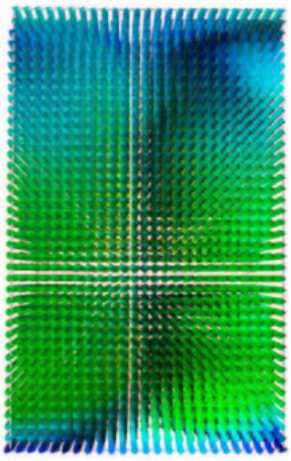
The construction of an object which can read and combine several data files, proves to be very useful in the construction of the visualization pipeline. Visualization methods only have to deal with the data itself, already combined, updated or subsampled when necessary. The global transformation matrix used by every visualization actor proves necessary in keeping all visualization objects aligned, while allowing the user to freely reposition and scale the flow representation.

The relatively large bounding box facilitates repositioning of the flow considerably. The anchor works well, but is less often used since it requires the user to navigate to the anchor before manipulation can take place. Experience with a large number of visualizations indicate that the flow representation is not constantly relocated or rotated, these actions are usually performed just after loading the data. As soon as the flow is positioned conveniently, it remains there until its destruction.

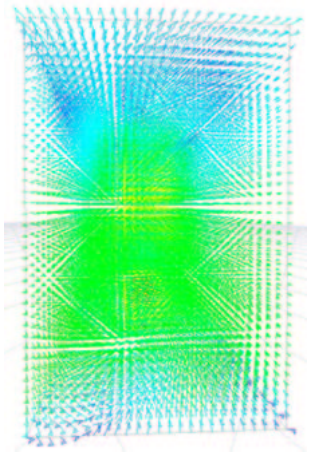
3.7.2 Results of global visualization methods

Glyphs

Glyphs are useful for locating singularities from a global viewpoint. Sudden changes in size or colour of neighbouring particles facilitate quick pin-pointing of these locations. Singularities are clearly visible by seemingly random behaviour of glyphs in a small area.



a: Cone visualization.



b: Hook visualization.

Figure 3.27: Glyph visualization of a whirlwind simulation without subsampling (see page 75 for a colour version).

Effectiveness of the visualization method

Glyphs were used to visualize the whirlwind dataset (see figures 3.9-b and 3.10-b). While rotating around the whirlwind or moving towards it, both hook and cone glyph representation showed the rotational patterns in the whirlwind. Cones provide a better sense of depth by their shading, but it helps to change the glyphs to hooks while navigating. A subsampling ratio is often necessary to clarify the visualization, since otherwise the screen becomes quickly cluttered (see figure 3.9-a).

When no subsampling is used, changing the glyph pointer from cone to a hook can improve the quality of the global visualization while allowing smooth interaction and navigation. A lot of cones can form a visual block by which the rest of the flow is occluded (see figure 3.27-a). It becomes difficult to see what lies beyond. The hooks create a kind of fuzzy visualization because each hook is mostly transparent. Figure 3.27-b shows the whirlwind without a subsample ratio. Areas with high or low scalar values are visible, even when they are located in the middle of the dataset, while the direction can be seen as well when the user is located close enough to the data. This kind of glyph visualization acts as a kind of volume rendering and renders relatively fast, thereby providing an interesting substitute for volume rendering.

Colouring glyphs is a useful technique to represent additional scalar information. Intuitive colourmaps ranging from blue to red are usually mapped on the range of values present in the flow-property . “Cold” blue objects indicate lower values and “hot” red objects indicate higher ones. To quickly understand the values of intermediate colours, a legend showing the applied mapping remains necessary.

Performance

The raw speed of the environment, without any visualization, provides a practical performance limit. A test application was implemented where the environment was rendered without any visualization method. A framerate of 185 frames per second was obtained, which can be used as the upper limit in further performance tests.

The raw speed of the visualization pipeline and graphics hardware was measured by visualizing several datasets with different subsample ratios from 81 to 1 with a step size of 5. Each step was visualized for 10 seconds, during which approximately 2 samples of the framerate were taken per second. A linear dependence between the framerate and the subsample ratio was expected because each glyph, either hook or cone, adds an equal amount of 3D primitives (vertices, polygons, normals and colours) to the final rendering.

Figure 3.28 shows the result. The lines show the mean framerate, the vertical bars depict the minimum and maximum framerate per second. Because the visualization pipeline updates when the subsample ratio is adjusted, the lowest framerate sample of each set was removed. Updates of the pipeline are likely to happen infrequently during normal interactive analysis, and therefore the effect is discarded.

When a large subsample ratio is applied, the framerate approaches the maximum of 185 fps fairly reasonable. The large differences between minimum and maximum framerates of the hooks visualization of both stenosis and Karman reactor datasets are most likely caused by the (non)availability of locks necessary to retrieve access between processes and graphics buffer-memory: the small amount of vertices makes a very fast rasterization possible, therefore most of the time the graphics environment is busy waiting for locks to become available. The differences diminish when the size of datasets increases, since they demand relatively much more visualization time.

The graphics card of the DRIVE system can process 31M triangles per second, and draw 1G pixels per second. These two limits do hardly restrict the tests: One frame consists of $1024 * ((768 * 2) + 40) = 1,613,824$ pixels, so the maximum number of frames is $1 * 10^9/1,613,824 = 619$. This will pose no limit on the performance. The number of triangles per cone is 22, so when the largest dataset, the double static mixer, is visualized without applying a subsample ratio, $699,608 * 22 = 15,391,376$ triangles are generated, which is half of the card's maximum; so a framerate of 2 can be obtained. If this would be the only practical limitation, applying a subsampling ratio of 10 would result in 20 frames per second, a reasonable performance for scientific applications.

An unsynchronized visualization environment is undesired though, since it wastes processor time by drawing frames which will not actually be seen. By synchronizing the visualization to the vertical screen retrace, the visualization is only updated while the monitor's cathode tube moves from the bottom-right to the upper-left of the screen.

The empty application was run while synchronizing the visualization to the vertical screen retrace. This resulted in a framerate of 67 fps which was to be expected since the vertical screenrate is set to 67Hz. This can be considered a practical upper-limit of the render speed that visualizations can reach.

During normal interaction, the user will often navigate around and through the dataset. To see whether this has any influence on the render time, the

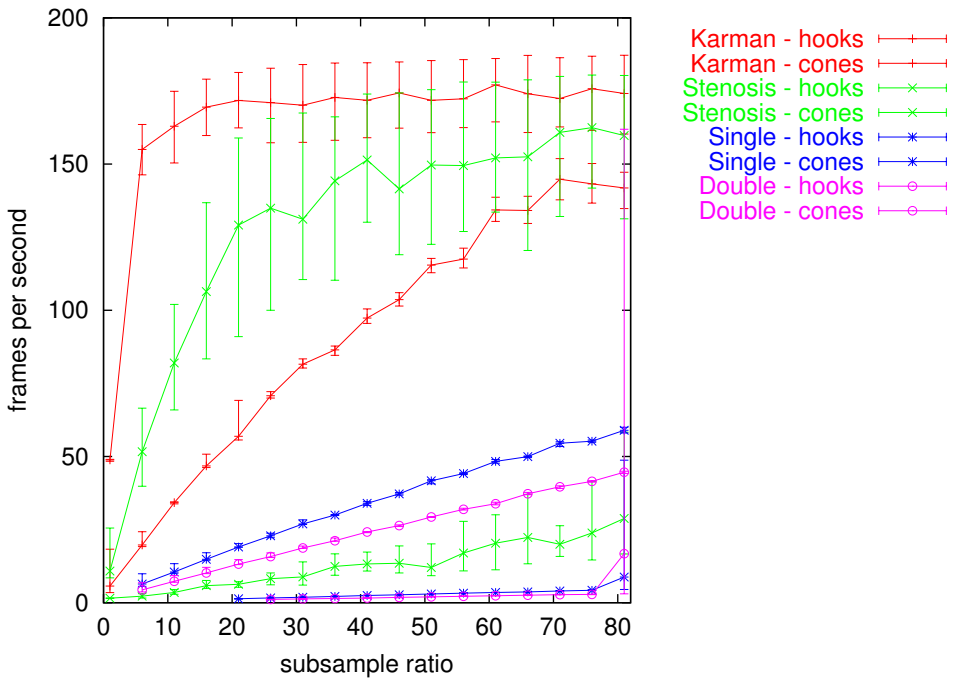


Figure 3.28: Glyph visualization of several datasets, not synchronized with the vertical screen retrace (see page 76 for a colour version).

stenosis dataset was visualized with different subsample ratios for 20 seconds. During the first 10 seconds the dataset remained motionless. It was animated up and down during the last 10 seconds in order to emulate navigation movement. Since the geometry of the visualization method does not change, the rendering process might speed up the creation of an image if data such as polygon-clipping is cached. When such methods are used, they will result in a considerable higher number of frames per second during static visualization.

Figure 3.29 shows the results. Again, the lowest framerate sample is removed, and mean, minimum and maximum values are shown.

The dynamic visualization of hooks is practically equal to the static representation. Cone visualization places a much heavier load on the graphical environment. The linear dependence between subsample ratio and framerate per second can be seen between subsample ratios of 1 and 30. The strange behaviour of the framerates between 31 and 81, as well as the influence of hooks and cones visualization on the render time, is discussed in the next test. An actual difference of almost 10 frames per second occurs between subsample ratios of 41 and 51, during the sudden "jump". Because the difference between dynamic and static representation is negligible on the overall domain, it can be concluded that navigation will not have a major impact on the performance of glyph visualization.

To get an overall impression of speed (render time) using glyph representation, several datasets were visualized with both hooks and cones, using different subsampling ratios from 81 to 1 with a step size of 5. When the total

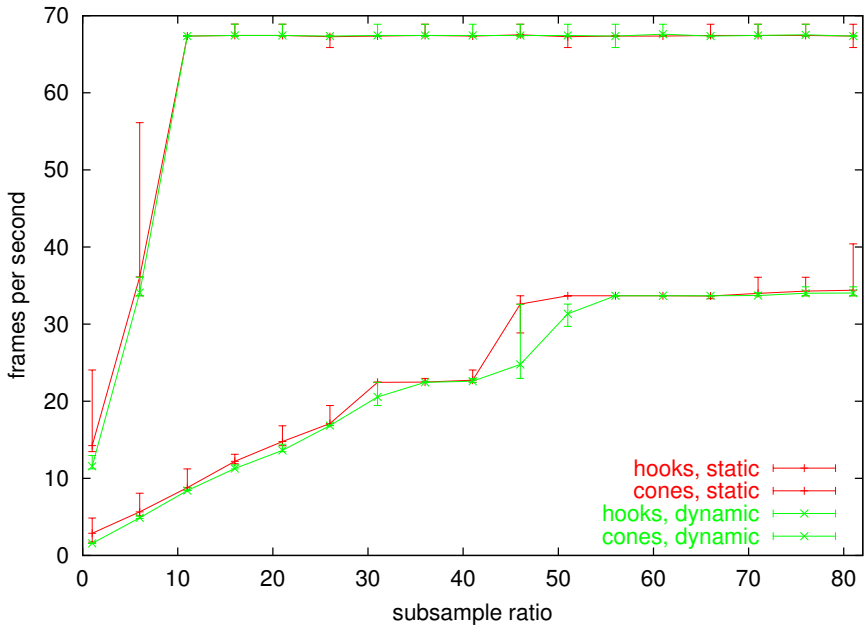


Figure 3.29: Glyph test depicting static versus animated visualization of the stenosis dataset, synchronized to vertical screen retrace.

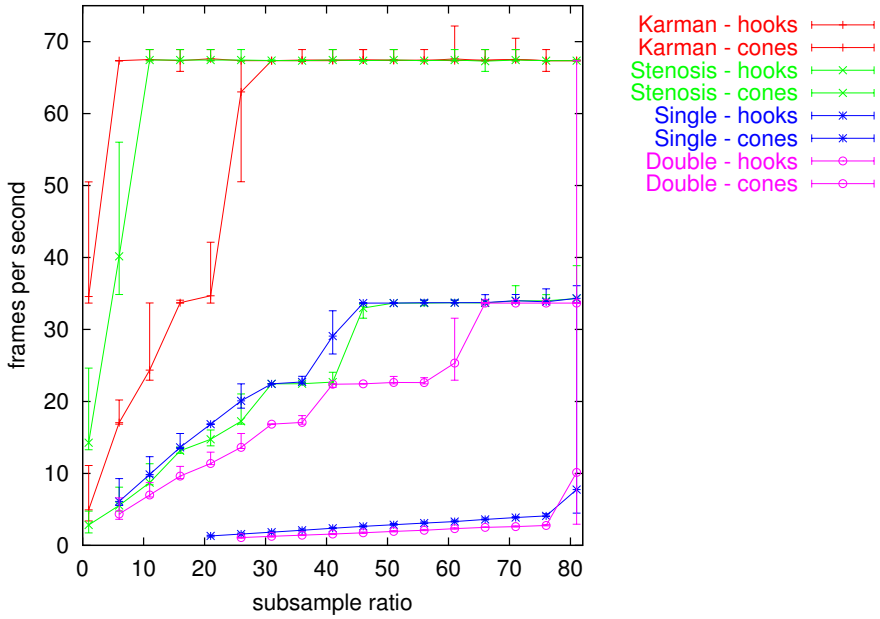
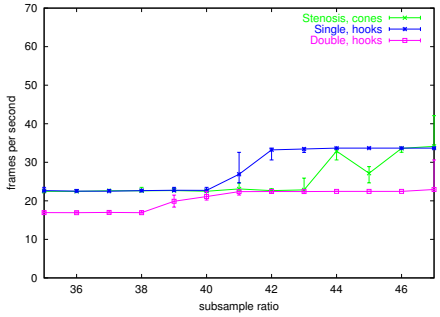
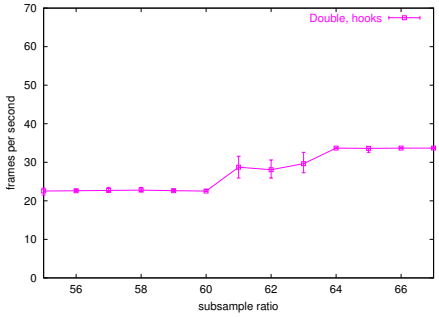


Figure 3.30: Glyph visualization of several datasets (see page 76 for a colour version).



Stenosis, single and double mixer.



Double mixer.

Figure 3.31: Hook and cone visualizations on subdomains using a step size of 1 (see page 77).

number of polygons or vertices crosses the maximum throughput of the hardware, a considerable drop in speed is expected.

The tests were synchronized with the vertical screen retrace, so the maximum possible framerate is 67 fps. The effect that during highly subsampled hook visualizations the framerate sometimes crosses the 67 fps boundary, is most likely caused by round off errors in the functions that calculate frame display time.

The linear dependence between subsample ratio and framerate of figure 3.28 seems only to apply to the large datasets of the single and double static mixer when visualized with cones. Between subsample ratios $36-46$ and $56-66$ during visualization of the stenosis (cones), single and double mixer (hooks) peculiar "jumps" occur, while the framerate remains quite stable before and after these subsample values.

A rerun of the tests with a step size of 1 was performed on these domains, shown in figure 3.31. The jumps hint at some kind of caching which is performed: such a drop in performance occurs when the amount of copied or swapped data crosses a certain threshold. Computing the amount of memory the visualizations required was fairly straight-forward in the case of hooks: each hook consists of 4 vertices, each vertex takes up 96 bytes of memory (3 coordinate floats, 3 color floats). Figure 3.32 shows an example of the memory size of single mixer and double mixer visualizations near the "jump"-subsample ratios. A threshold of 1,048,576 bytes (1024^2) would be logical but does not appear to be present. The multiple stages of the rendering process, which includes the visualization pipeline, copying of shared memory segments and data-transfer to the graphics card, makes it difficult to find the bottle-neck which is causing this behaviour. Since the graphics card specifications seem to be quite able to handle these amounts of data (see the start of this section), the bottleneck will probably be present in either the VTKtoCAVE library or the visualization network.

The results show that only glyph visualization using hooks can guarantee a smooth framerate of at least 15 fps using reasonable sampling rates. The cones demand a lot of computing power from the graphics card and can therefore only be used when a high subsampling ratio is applied. Whether this is feasible depends on the flow data and the kind of analysis the user is performing.

$Single_{40}$	= 11,725 glyphs	= 1,125,600 bytes
$Single_{42}$	= 11,166 glyphs	= 1,071,936 bytes
$Double_{60}$	= 11,661 glyphs	= 1,119,456 bytes
$Double_{61}$	= 11,469 glyphs	= 1,101,024 bytes

Figure 3.32: Memory size of several subsampled datasets visualized by hooks.

Raycasting

Effectiveness of the visualization method

In figures 3.33 and 3.34 the stenosis dataset is visualized by raycasting. Underneath each representation, the applied lookup table is shown. The lookup table shows a histogram of all scalar values present in the dataset from left (minimum value) to right (maximum value) , denoted by the light-gray vertical lines. The coloured circles on the black line show the mapping of the scalar values to colour. The vertical axis represents the mapping of scalar values to transparency: from the bottom, completely transparent, to the top, completely opaque.

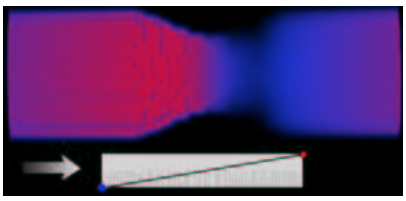
The expected speed-up of flow in the middle of the stenosis (figure 3.33) can clearly be seen. By mapping high speeds to colours such as red and yellow an intuitive picture of the behaviour of the scalar flow property is obtained.

a: Front view *b*: Perspective view.

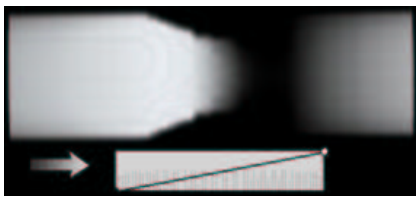
Figure 3.33: Stenosis speed visualization. The arrow denotes the flow direction (see page 77 for a colour version).

In a stenosis, a drum-shaped pressure-profile is expected: the build-up of pressure when the tube narrows will be the highest in the middle of the artery following the bloodflow and it will decrease towards the edge of the artery. A colour mapping from blue to red does not visualize this very well (figure 3.34-a). The human eye is much more sensitive to the colour red than to the colour blue, causing a kind of "over-exposure" which makes the exact pressure hard to see. The sensitivity to gray values is much higher compared to colours, therefore a non-linear black-white lookup table will accentuate the pressure-profile much better (see figure 3.34-b).

This example shows the importance of access to a flexible lookup table when using raycasting methods. The scientist will never be able to choose the perfect lookup table beforehand, so easy interaction methods through which he can make adjustments must be available.



a: Blue-red lookup table.



b: Black-white lookup table.

Figure 3.34: Stenosis pressure visualization. The arrow denotes the flow direction (see page 77 for a colour version).

Performance

The necessary computations require a lot of time and therefore a low-resolution version of the image was used while its lookup table was adjusted or when it was moved or rotated. Exact figures are not available, but a delay of at least 4 to 10 seconds while changing from low-quality to high-quality representation was not uncommon. This can provide difficulty in a VR environment. Noise from the tracking system and small movements by the user's head can constantly trigger updates of the visualization, slowing down interaction time considerably.

Iso-surface

Surface rendering proved to be very useful concerning the visual geometry representation. The bounding box of `cFlowField` only shows the outlines of the dataset without any hint concerning the geometry. An iso-surface allows a simple but effective outline representation of the geometry (see figure 4.8). A detailed outline is not necessary so the number of polygons can be kept relatively low, allowing smooth rendering.

The visualization of voxels with a data-property, such as speed or pressure, above a certain threshold provides a useful representation. When several surfaces are created with evenly spaced iso-values, the distance between subsequent surfaces visualizes the gradual change of this flow property. Where surfaces approach each other the change appears rapidly, while widely spaced surfaces indicate a slow increase or decrease.

Performance

The response time, the time necessary to compute, decimate and triangulate the surface, takes usually at least a couple of seconds which is relatively long. To reduce the time spent on trial-and-error while searching for useful surface representations, interaction methods such as histograms should be present which facilitate the choice of iso-values.

Exact figures of performance are difficult to compare because the resulting amount of polygons depends completely on the used flow data and chosen iso-value(s). In general, surfaces place a heavy load on the performance and they often slow down render times considerably. Decimating the surface will therefore always be necessary, where the percentage of decimation with regard to the surface depends on the size of the flow data.

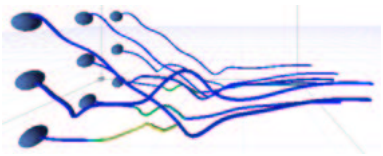


Figure 3.35: Converging streamlines in mixer.

When an iso-surface is used to show the geometry boundary, which does not require a high-quality representation, considerable speed-up can be obtained by visualizing it as a wireframe instead of a smooth shaded surface.

3.7.3 Results of local visualization methods

Streamlines

Figures 3.35 and 3.36 show streamlines in a dataset with a fairly complex geometry (see figure 3.26). The swirling ribbons of streamlines give a good insight in intricate flow-motion. Converging and diverging streamlines help to understand the movement of flow around obstacles. During global review of the flow, streamlines are less useful because a large amount of streamlines is necessary to fully cover a flow area: computation time increases and the number of framerates per second drop considerably. When the location of a singularity is known, streamlines are a valuable tool to investigate them in detail.

Effectiveness of visualization method

Colouring streamlines with a scalar works well. Streamlines are useful to study the dynamics in both direction and scalar flow property. Shaded lighting is preferable since it provides a better depth-cue of the tube. Visualizing the rotation of the tube was not particularly successful: the rotation can often hardly be seen.

The goal of the single mixer is to mix two viscous fluids which enter the mixer on top of each other. The ripples of the streamlines show their swirling motion around the mixer's geometry. Streamlines which enter the mixer at high and low positions should be "mixed" when they leave the mixer. Figure 3.35 shows streamlines originating from evenly spaced cursor points, which converge through the mixer towards the right side. They are coloured by speed scalars and show no extreme speeds. It also shows the danger of using too many streamlines: when they constantly converge and diverge, it might not be clear what the general movement of the flow is. In that case, it might be best to reduce the amount of streamlines, perhaps increase the distance between the cursor spheres, and analyze the resulting streamlines. By moving the cursor around, the scientist can check whether this behaviour is common for the complete flow or whether irregularities occur at certain areas.

Performance

Measuring performance of local visualization methods such as streamlines is difficult, since the visualization and the resulting amount of polygons depends

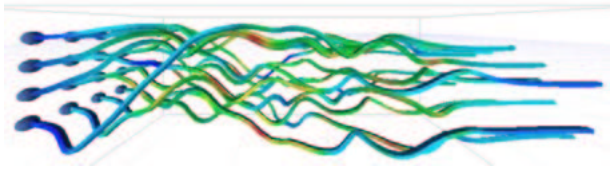


Figure 3.36: Converging streamlines in double mixer (see page 78 for a colour version).

on the location of the cursor or visualization parameters in combination with the dataset. The difference in length, computation time and resulting points of two streamlines originating from different locations can be considerable. They depend completely on the flow properties so no comparative results can be obtained by using several datasets with a local visualization.

Though streamlines consist of a relatively low amount of polygons the required render time can still be considerably. While positioning the streamline, a faster update and visual feedback is preferred. Tests with several datasets showed a considerable speedup while visualizing the streamline by a line representation. When the cursor is released, a small delay in feedback is experienced but this proved to be acceptable. The technique of increasing the step size (resulting in a more rough visual representation of the computed streamline) while the cursor is being dragged, combined with a visually less demanding representation, works well. But it is mostly the absence of tubes which speeds up of the visualization. Increasing the integration step to decrease computation time is not very effective since the visible streamline at the moment of dragging might be incorrect because of overshooting and the computation time is relative small compared to render time.

Animated particles

Animated particles provide useful clues in areas with complex flow movements and high speed differences. Releasing several particles from nearby locations shows the speed with which the flow diverges and converges. High or low velocities are often points of interest; the change in speed of particles visualizes this in an intuitive way.

The mapping from a scalar flow property, such as pressure, to colour or size as used by glyphs was included in the particles and helps to provide additional information. When using animated particles, they sometimes move too fast to provide a good estimation of the property at their position. But while using static particles, they provide useful information concerning the specific flow property.

The static particles were used to visualize the whirlwind dataset (figure 3.38). The visualization shows that particles near the border of the flow converge towards the nexus which goes almost straight upwards. Towards the upper boundary of the whirlwind, the particles slowly diverge again.

The particles are separated by a fixed time period. The speeding up (near the middle) and slowing down (near the upper and lower boundary) of particles is visualized by decreasing and increasing distances between subsequent particles. This improves the quality of the flow's visualization compared to glyphs and

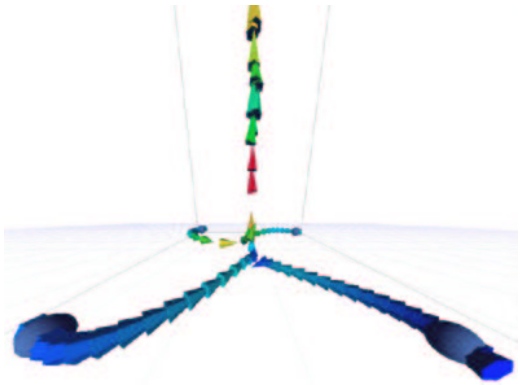


Figure 3.37: Close-up of particles entering the nexus of the whirlwind (see page 78 for a colour version).

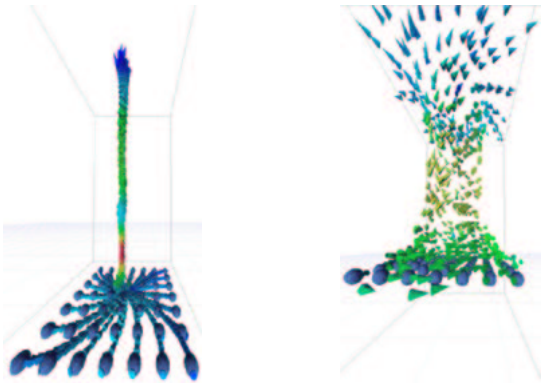


Figure 3.38: Particles entering the whirlwind at different heights (see page 78 for colour versions).

streamlines considerably.

3.8 Discussion

Depending on the goal of the research, scalar and vector flow properties demand different types of visualization to aid during analysis. By providing different visualization methods with distinctive features, analysis possibilities are improved by visualization the flow in an immersive environment.

The parameterization of visualization variables is a difficult step in the visualization process. The discussion section of the test case chapter describes this in more detail.

The qualitative and quantitative measurements of FlowFish's visualization classes as described above provide only partial test results. The tests showed the possibilities and advantages of each method with regard to the data set which was visualized. A more general test of visualization methods would use comparable sets of flow data with each visualization methods. By comparing the same visualization method with several data sets, more information regarding

the advantages and possibilities of a visualization method can be obtained. Furthermore, by comparing different visualization methods on the same flow data and letting experts use the methods to analyze the flow, the qualitative results of these methods can be compared better. These comparisons can be used to suggest specific visualization methods and parameterizations for specific flow data or visualization purposes.

The performance tests gave indications of possible bottlenecks which might affect a smooth interaction. Quantitative measurements obtained from a wide range of flow data can result in more general conclusions concerning the possible problems in response time and framerate of visualization methods.

Using widgets such as `vtkSourceCursor` provides an intuitive manipulation of 3D objects, therefore these methods should be used and extended for the purpose of controlling visualization parameters.

Chapter 4

Test case: Flow visualization in a simulated vascular reconstruction environment

As a test case, the FlowFish application is used in the Simulated Vascular Reconstruction Environment (SVRE) project [55]. The aim of this project is to provide a surgeon with an intuitive environment to visualize and explore a patient's vascular condition. By placing him in an interactive virtual simulation environment, a surgeon or radiologist can examine the patient's bloodflow. Besides aiding the scientist in his research, the usage of virtual simulation and visualization techniques aims to overcome practical problems present in a real-life situation. The environment enables the employment of investigation methods which cannot be used in the original environment of the flow. To test a hypothesis, different surgical procedures can be applied whereupon the surgeon can monitor direction, speed and pressure of the bloodflow through the human vascular system.

From a medical point of view, the SVRE project focuses on problems related to bloodflow through the arteries and veins. Part of a patient's artery is scanned with a Computed Tomography (CT) scanner or by Magnetic Resonance Imaging (MRI). The 3D dataset is loaded into a virtual environment. This environment does not only allow a close examination of the data, but also provides the surgeon with virtual tools to change the geometry of the artery, for example to simulate a bypass. The new geometry is sent to a supercomputer which computes the bloodflow through this virtually reconstructed artery. By updating the flow while subsequent time steps are computed, the surgeon gets immediate visual feedback and, when necessary, can adjust the geometry again, stop the flow simulation and restart it until a suitable treatment has been found.

From a computer science point of view, the SVRE project addresses several problems: the parallel simulation of flow resulting in a correct simulation of real blood, the effective transfer of vast amounts of data over a network from a simulation environment to the representation environment, advanced visual

representations and intuitive interaction and the scientific issues involved in these interactive dynamic exploration environments [55]. The visualization library presented in this paper provides a solution to the visualization part of the SVRE-project. It strives to present the surgeon with a powerful set of tools and visualization classes which are both intuitive to use and flexible enough to obtain a clear understanding of the information contained in the datasets.

4.1 Vascular disorders

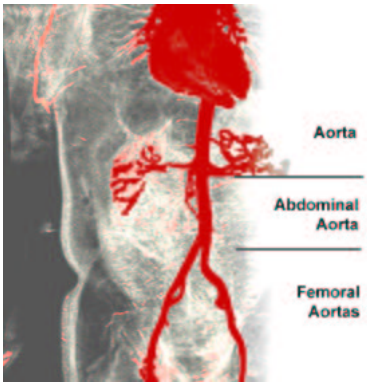


Figure 4.1: Volumetric scan of the abdominal aorta bifurcating into the femoral aortas.

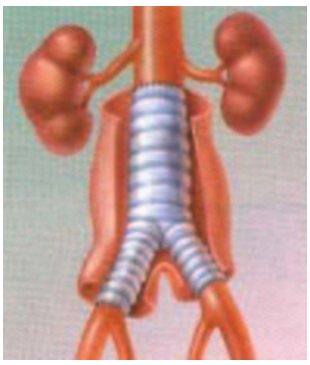
Veins and arteries are flexible tubes that transport blood through the body. The arteries transport oxygen-rich blood from the lungs, via the heart, towards the organs. Veins carry the blood, from which the oxygen is taken, back to the lungs. The aorta (an artery) is the largest blood vessel. When it passes through an opening in the diaphragm it becomes the abdominal aorta (figure 4.1). Near the waist, the abdominal aorta bifurcates in the left and right femoral arteries, into both legs.

Vascular disorders can take place in arteries as well as in veins. Depending on the location and size of the disorder they can be life-threatening. Their cause can originate from a number of situations, such as genetic disorder or bad nourishment. With both stenosis and aneurysms, the geometry of the artery is of major importance while deciding upon treatment and severity. Classical visualization methods provide the surgeon with 2D slices of the region, taken by an MRI- or CT-scan. He has to construct the "real" 3D geometry in his mind while scrolling through the images. This method is vulnerable to errors, due to the overlooking of details or an incorrect reconstruction in the surgeons mind. It takes quite some time to learn this technique and even experienced surgeons can overlook problem areas, for example when these are aligned with the slicing direction.

Aneurysm

An abdominal aortic aneurysm (AAA) represents a localized dilation of at least 29 mm of the largest blood vessel in the lower part of the body, which supplies

a: Abdominal aorta with aneurysm.



b: Placing of Dacron graft.

Figure 4.2: Abdominal aorta aneurysm.

blood to the legs (figure 4.2). The frequency of aneurysms to form in this location relates to abnormalities in blood flow as well as changes which occur to the blood vessel wall over time due to hardening of the arteries (atherosclerosis). 5 to 8 of every 100 men over 65 years old are affected by AAA, and about 2 of every 100 women. In general, aneurysms expand over time and ultimately burst, much like a balloon which continues to expand and finally blows up. The rupture has catastrophic consequences 85% of the time. Only 18% of all patients with ruptured AAA reach a hospital and survive surgery [56].

Treatment Treatment depends on the size and location of the aneurysm and overall health. Patients with small (< 4-5 cm.) or stable aneurysms in the descending aorta or abdominal parts (those farthest from the heart) usually have regular check-ups and can live with the aneurysms for years. Doctors may prescribe medicine to lower the blood pressure, thereby relieving the stress on the artery wall. When the aneurysm reaches around 5 cm. in diameter, there is a sharp increase in chance of aneurysm rupture and surgery is recommended [57]. Without major complications, the procedure consists of the following steps:

- The aorta above and the common iliac arteries below the aneurysm are clamped off with special vessel clamps;
- the aneurysm is opened and any blood clot (thrombus) on the inside of the aneurysm is removed;
- any small artery openings in the aneurysm wall are oversewn;
- a prosthetic graft made of Dacron (which causes no reaction by the body) is sutured in place between the two ends (figure 4.2);
- the aneurysm wall is then wrapped around the Dacron graft;
- the clamps are removed and the wound is sutured.

Stenosis

Stenosis is a narrowing of the artery. This constriction can be caused by the build-up of fat, cholesterol and other substances over time. If the narrowing is

severe, it reduces the flow of oxygen-rich blood to the body, which can result in hypoxia (shortage of oxygen) and lung congestion. At the same time, the heart has to work harder to push blood through the narrowed passage and may become damaged as a result. 87% of all deaths and 74% of all hospitalizations related to stenosis occur in persons of age 65 years or older [58].

Treatment There are different options for treatment, depending on the severity of the patient's condition. When the artery-wall is strong enough, and the build-up not too large, the artery can be opened and the surgeon can manually clean the inside of the wall. Another option is balloon angioplasty: a balloon-like object is inserted into the artery. By pushing slowly, it is positioned right in the middle of the stenosis, and slowly pumped up. The "balloon" is filled and pushes the wall away, thereby widening the artery.

After a successful operation, the patient must visit the hospital frequently. With periods varying between a few weeks and a couple of months, the artery must be checked for evidence of stenosis, especially near the region where the previous stenosis took place. The build-up often re-occurs at the same place or region, which is more vulnerable.

4.1.1 Scientific visualization in medical analysis

There are situations in which the possible course of action is not clear to the surgeon. Treatments are not without risk, and must be avoided as much as possible. Especially when surgery must be performed, it is not always clear how to perform the operation and what the influence on the bloodflow will be. The altered bloodflow depends heavily on the exact location and method of the surgery. An environment in which a real-life situation can be simulated and several treatments can be tested, will provide useful clues to the surgeon. A number of criteria must be met for such an environment to be useful:

1. Patient data. The surgeon must be able to feed the environment real data. A patient's artery must be scanned and digitized in sufficient detail as to enable simulation.

2. Accurate simulation methods. If a surgeon needs to simulate a treatment, the computer must be able to simulate blood running through the vein accurately. It is unthinkable to let a surgeon make a decision upon computer results when it cannot be guaranteed that they are accurate and simulate the bloodflow realistically. Furthermore, the simulation method must be fast enough to be of any use. When the simulation takes weeks or even days to finish, its result will be useless. This criterium can be relaxed when the environment is solely used for inspection of a patient's condition.

3. Simulation result presentation. The data must be presented to the surgeon in such a way as to assist the surgeon in making the choice of treatment of the patient's condition. The presentations should be intuitive and easy to understand, without the need of extensive training.

4. Intuitive interaction. Facilitating a quick and powerful inspection of the medical data, the surgeon should not be hampered by poor interaction methods. Positioning oneself in a virtual environment might take some time to get used to, but ideally it should be as intuitive as navigating in the real world.

Imaging techniques such as X-ray angiography, MRI or CT scanners can be used to scan a dangerous region, which in turn can be diagnosed by a physician by means of a virtual environment to see if there is a chance of a distortion or possible disorder. Further details about this research can be found in [55].

4.2 Results

Any major irregular phenomenon of a bloodflow through an aneurysm or stenosis, like a sudden change in speed, direction or pressure, backflow or stagnant flow is of importance to the physician in order to make a reliable choice concerning the treatment. The visualization techniques described in section 3.5 must aid in locating such irregularities and, as far as the bloodflow concerns, the cause.

The visualization methods were tested on the DRIVE system (section 2.3.1) with 2 medical datasets:

- The stenosis dataset described in section 3.7.2.
- Bloodflow through the abdominal aorta was simulated using the D3Q19 quasi incompressible LBGK method [59]. The time-dependent snapshot is taken from the systole, where pressure builds up (see figure 4.3). The simulation was created using Reynolds number $Re \simeq 1150$ and Womersley parameter $\alpha = 16$ which is common for blood during rest periods of the human aorta. A speedup in the femoral arteries is expected, with the highest velocities near their axis. They were created by A.M. Artoli using a dataset from the Stanford University [14].

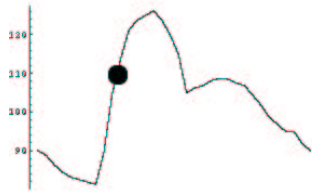
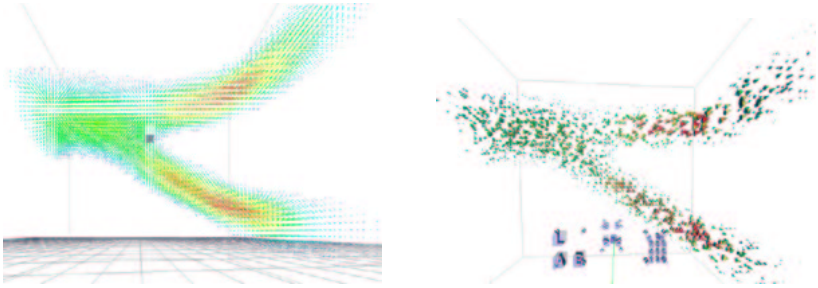


Figure 4.3: Pressure (y) versus time (x) of bloodflow through the abdominal aorta during a systolic cycle. The black dot denotes the snapshot which is used in this test case.

The interaction classes `vtkCAVEMenu3D` and `vtkSourceCursor` and the general purpose class `cFlowField` function similarly with medical datasets as they do with other data. Therefore no additional tests were necessary.



a: Hooks and no subsample ratio. *b*: Cones and random subsampling, with flowfield and glyphs menus.

Figure 4.4: Abdominal aorta (see page 79 for enlarged images).

Glyphs

Both cones and hooks create representations useful for flow analysis. The hooks need no subsampling while still allowing a smooth interaction with the data (figure 4.4-a). Because of the high density the orientation of glyphs is mostly lost and it remains difficult to find the general direction of the flow or anomalies. But when the dataset consists of such a large amount of voxels, the glyph-hooks can actually be used as a kind of rough raycasting method. As in raycasting, the direction of the flow is not obvious anymore, but colouring the glyphs by a scalar flow property creates the "fuzzy" visualization which is expected by this abdominal aorta flow simulation. This depends heavily on the relative amount and placement of non-zero vectors in the dataset.

A subsampled collection of cones does provide more information about the direction, while still showing a good global view (figure 4.4-b). Areas with higher speeds are filled with larger glyphs, immediately drawing attention to these areas.

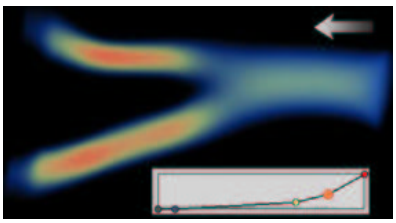
a: Normal subsampling. *b*: Random subsampling.

Figure 4.5: Stenosis visualization.

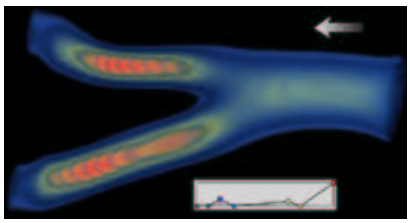
The stenosis dataset provides a good example of the necessity of random subsampling (figure 4.5). With an unfortunate subsampling value, anomalies or interesting regions might quickly be overlooked. A practical problem concerning both datasets were the extremely small vectors. The scaling of glyphs by vector magnitude works well, but in order to visualize a large group of glyphs with relative small sizes, the anomalies might obscure part of the data (figure 4.5-b).

Raycasting

The dataset of an abdominal aorta (figure 4.6) was visualized on the VolView application.



a: Standard mapping.



b: Highlighted mapping.

Figure 4.6: Front view of flow velocity in abdominal aorta (see pages 80-81 for enlarged and perspective views).

Figure 4.6-a uses an intuitive mapping from blue to red, resulting in a visualization where the interesting (high flow speed) areas can be located. When for example a specific threshold of speed is important, the mapping can be refined to highlight the visualization of these areas (figure 4.6-b). The dark ring around the high-speed area's seems to indicate a sudden drop in velocity which is not present in the actual data (figure 4.6-a). The spikes in the lookup table cause this phenomenon, in order to highlight the high-speed area. Care must be taken when such a mapping is used for datasets with unknown behaviour, in order not to create phenomena in the visual representation which are solely caused by the choice of mapping values.

These 2D examples of raycasted images show that volumetric rendering is a very powerful method in flow visualization. They provide an intuitive representation of its scalar properties such as pressure or speed. The VolView application uses a low resolution image during manipulation of the visualization. When the object is released, a high resolution image is computed, which usually takes only a few seconds. Thus, smooth interaction and high quality images are successfully combined.

Surface rendering

Medical datasets describing arteries and veins often contain irregular geometries. Finding the actual flow can take valuable time. Without an artery-wall representation to locate the actual flow, the user might move the cursor through zero-speed voxels, which will not create any visual representation. Surface rendering can create a simple wall representation, aiding the user in positioning the cursor immediately inside the flow.

Surface rendering proved valuable in highlighting areas above a specific pressure or speed (see figure 4.7). Often positioned deep within the bloodflow, the surfaces provide clear representations of these areas. The visualization of low-pressure areas was not very successful with the used datasets. They did not contain any irregular low-pressure areas deep within the flow. Low speed and pressure values were mostly found near the artery wall. A surface visualization of these areas creates a kind of "second skin" near the geometry surface, without adding distinctively useful information. But in other medical datasets the occurrence of irregularities such as a low-pressure area located deep within the artery is very well possible. Surface rendering will help locating these areas quickly.

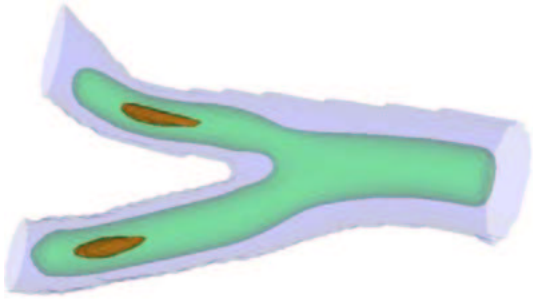
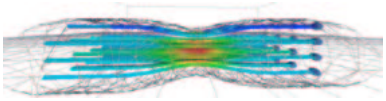


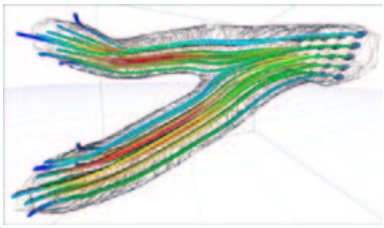
Figure 4.7: 3 iso-surfaces depicting the geometry of the abdominal aorta, mean (green) and high (red) speed visualization.

Streamlines

Figure 4.8-a shows the convergence of the stream in the stenosis. The colour of the streamtubes shows the speed, using a lookup table as shown in figure 3.8-b. When encircling the visualization, the symmetry around the stenosis' axis can be clearly seen.



a: Streamlines, originating from a plane cursor, integrating through a stenosis.

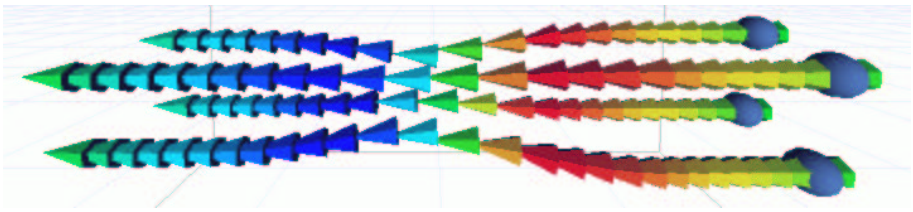


b: Perspective view of aorta.

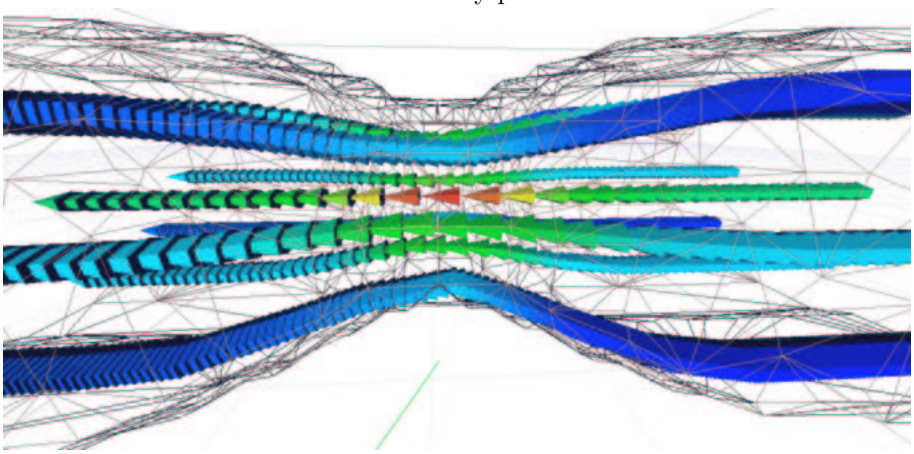
Figure 4.8: Streamlines in medical datasets (see page 82 for enlarged images).

The extremely small vectors often present in medical datasets provide a slightly more difficult task for the surgeon while fine-tuning the visual representation. A correct integration-step might take some time to find. The terminal speed needs careful adjustment as well. If it is set too high, streamlines might be cut off prematurely or not be able to start integrating from the cursor at all. If it is set too low, the complete virtual environment can be slowed down due to the computation and creation of thousands of streamline-points packed close together.

The expected increase in speed in both femoral aortas is visualized by colouring the streamlines by the vector-magnitude (see 4.8-b). As expected, the increase in speed is the highest in the middle of both femoral aortas, decreasing towards the wall.



a: Coloured by pressure.



b: Coloured by speed, with artery wall representation.

Figure 4.9: Close-up of particles through a stenosis (see page 83 for a colour version).

Animated particles

In figure 4.9-a the cursor is placed on the right side of the artery, symmetrical around its axis. The particles show the convergence and speedup near the narrowing of the tube. They are coloured by the pressure of the flow by using a blue (low) to red (high) lookup table. The expected increase in pressure near the narrowing of the artery is visualized by the red glyphs, which drop back to blue as soon as the particles are pushed through the stenosis. The small vectors, located on a very small domain, posed no problems for the lookup table, since the animated particles were only positioned on voxels which contained values within this subdomain. Figure 4.9-b shows a similar snapshot, but here the particles are coloured by their speed. The wireframe representation of the artery wall facilitates the positioning of the cursor.

4.2.1 Discussion

The test case used 2 medical datasets visualized by FlowFish. They show that the FlowFish library has potential for visualizing such complex flows in an immersive environment. Such an environment provides many clues and hints to the surgeon, extending the limited representations of 2D environments by far. Still, a lot of work has to be done before doctors and surgeons can take benefit of these visualization methods.

The parameterization remains difficult. During this research, “correct” set-

tings were only empirically obtained through extensive use of such datasets. When a "correct visualization" is defined for a specific analysis such as surgeons need, it can be used to set visualization parameters automatically wherever possible. The remaining parameters must be translated to a definition the end-user (e.g. a surgeon) is familiar with, so he can use them intuitively.

The interaction between user and visualization object is yet far from intuitive. Widgets like histograms must be available which show statistics of flow properties such as minimum and maximum values. Using these, the doctor can find interesting regions faster.

Chapter 5

Conclusion

This thesis presents a flow visualization library, FlowFish. Its goal is the creation of visual representations of complex flow data in virtual environments to aid in flow analysis. By immersing the scientist in a virtual environment, FlowFish offers direct interaction with the visualizations and their parameters. This facilitates intuitive navigation and interaction and provides additional guidance over desktop visualization environments by means of depth-cues such as motion-parallax and stereoscopic images.

FlowFish includes support classes for data-handling methods and interaction, global visualization methods (such as glyphs and iso-surfaces) and local visualization methods (such as streamlines and animated particles). The scientist is able to acquire both a global as well as a detailed view by combining these various methods.

The visualization methods can combine both vector and scalar properties of flow into their visual representation, thereby clarifying the mutual relation between flow properties and their behaviour over time. FlowFish allows the update of time-dependent flow visualizations as soon as an update of data occurs. Therefore changes in the flow are immediately reflected in the visualization environment.

Visualizations do not show the exact values of flow data. Instead a mapping from data values to visualization parameters is used. Flow data is represented by geometric constructs that are represented visually. This relative representation does not pose a real limitation on the visual representations; often the scientist is interested in the relative behaviour of flow properties, e.g. relatively low speed or relatively high pressure. The interaction items associated with the visualization parameters therefore refrain from using absolute values as much as possible since the relation between an absolute value and a relative visual representation might not always be clear. But when in-depth knowledge about the flow data is available, some probing tool which shows absolute values, can be useful [32].

Because the flow visualization methods presented in this thesis are used in interactive VEs, a trade-off between visual accuracy and visualization speed is often necessary. Performance tests show that sometimes relatively large amounts of data must be left out of the visualization in order to maintain good performance. FlowFish's classes include several methods to reduce complexity and increase performance, such as reduction of data and simplified visual represen-

tations, offering both complex visualizations as well as smooth interaction.

In order to obtain the highest possible image quality, a good parameterization of visualization parameters must be found. Even experienced users have difficulties with finding useful values while visualizing an unknown dataset. There is not much the visualization application can do but to offer clues by means of small visual data-analyses such as histograms. The "useful" value of visualization parameters in terms of effectiveness and performance, depend heavily on the represented data. Methods to analyze the data and automatically find useful settings for a wide range of datasets, proves a very difficult task, one which is not solved in this project.

5.1 Discussion

The tests were performed with only a few flow datasets. To compare the visualization methods unbiased, more data is actually needed. Experts in different scientific fields concerning flow simulation and analysis should give their opinion on how well the visualization methods work by judging the resulting visual representations of both known and unknown flow data.

Finding the correct parameterization of visual representation remains a complex subject. By lack of expert systems, human intervention remains necessary. There are many settings though, which have a common dependence on the data: for example, when the properties of the hardware graphics engine are known, a visualization instance can choose to apply a certain subsampling ratio depending on the size of the original data [60].

Most flows encountered in real life are not stable. Independently of the applied visualization method, great care must be taken while analyzing the flow by means of one or more snapshots. Whether a single snapshot can be held representative for the overall flow behaviour over time, is a question which must be considered carefully for each flow with regard to the goal of the analysis. The perfect solution would be hardware which is fast enough to render scientific visualizations real time, offering both single snapshots as well as a continuous representation of time-dependent flow.

At the current state of hardware, such fast visualizations are not yet possible. An approximation of the overall flow behaviour might be created by showing several snapshots or compute a kind of "overall flow state" from different snapshots. But both a single snapshot as well as an overall view of the data suffer from "focus" problems; a snapshot disregards both previous and future behaviour and anomalies, while the overall view disregards anomalies occurring in a single time step. Because hardware and software are as of yet not fast enough to render a complex flow in a real time immersive environment without omitting a significant amount of data, care must always be taken while analyzing the representations. The scientist must never confuse the representations of data for the real data.

The flow visualization presented in this thesis show a promising trend in performance and picture quality. The presented visualization methods all have their advantages and will create a useful visual representation to scientists when combined and shown in an immersive virtual environment. Speed and performance of hardware keeps expanding, offering faster visualizations and computations every year. When high-quality visualizations of complex flow can be visualized

on a commodity workstation today, wonderful visualization possibilities will become possible in the near future.

5.2 Future work

The creation of intuitive interaction objects in an immersive virtual environment still remains a challenge. Useful widgets such as sliders and lookup table representations, which should be relatively small and easy to use, will greatly improve the interaction possibilities. The VTK library upon which FlowFish is built, includes items such as scalar bars, which show the mapping of a flow property to visualization parameter. Unfortunately, the VTKtoCAVE library does not support these objects in the DRIVE environment yet.

Experiments with raycasting methods show great potential. Work should be done to include this visualization method in the DRIVE environment using hard- and software support.

Recent ideas for different visualization methods which might help to visualize anomalies such as backflow will extend the visualization and analysis possibilities offered by FlowFish. Related work recently published by others show alternative flow visualization methods that could be interesting [24][25][61].

The importance of time-dependent flows in flow analysis demand an interface that facilitates access to different snapshots and enables playback, stop, rewind and synchronization options (for example, with animated particles).

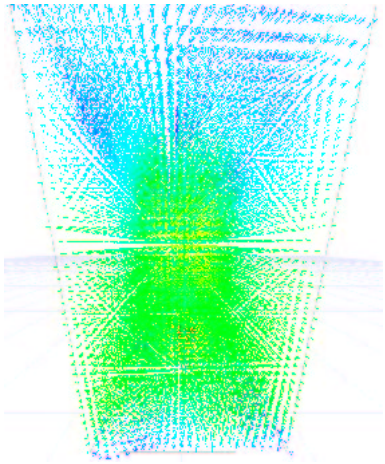
Navigation through an environment such as created by FlowFish is often centered around only one visual object. The user often zooms in or out while circling around the object. Instead of a common "Eulerian" navigation, a polar navigation most likely decreases navigation-time. The user-movements describe a sphere with the visualization object as its center.

Automatic adjustments in order to find the best combination of image quality and performance might be performed by FlowFish, visualization objects or the visualization environment, but this is far from trivial. There is an ongoing research in the section Computational Science of the UvA concerning the implementation of intelligent agents which can be used to set visualization parameters (semi-)automatically.

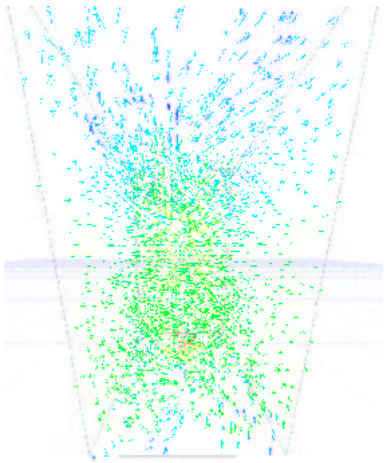
More tests can be performed on the performance of visualizations, in order to locate bottlenecks. The multiple stages in the visualization process turn this into a complex analysis which is far from straightforward. Some initial testing has been performed on glyph visualization, but a more in-depth analysis of glyphs and other visualization methods might reveal interesting results and behaviour of the different parts of the visualization pipeline.

Appendix A

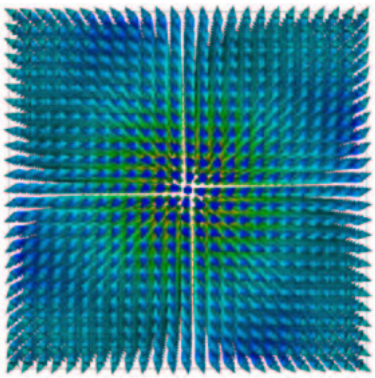
Colour plates



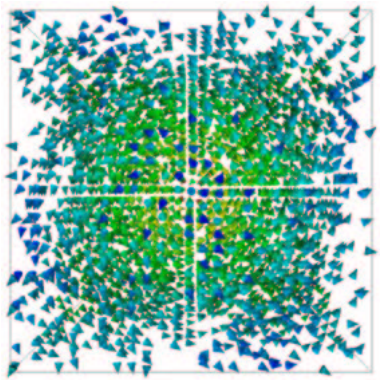
a: Original dataset (hooks, front).



b: Subsample ratio of 8.

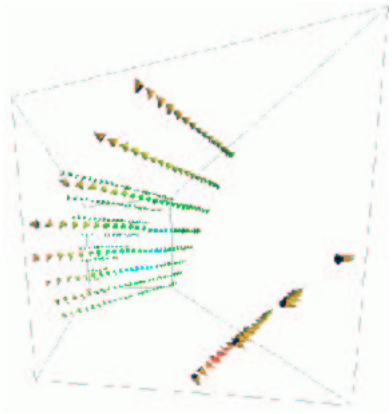


c: Original dataset (cones, bottom).

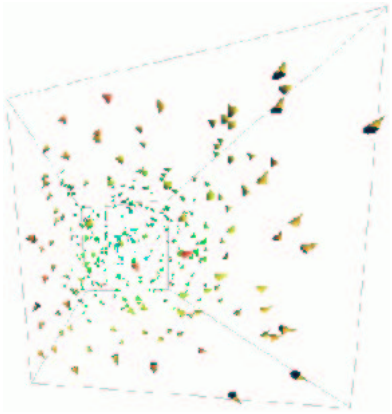


c: Subsample ratio of 7.

Figure A.1: The effect of subsampling on glyph visualization of a whirlwind simulation.

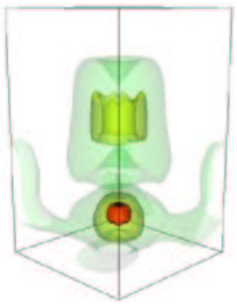


a: Aliasing effect in a regularly subsampled dataset.

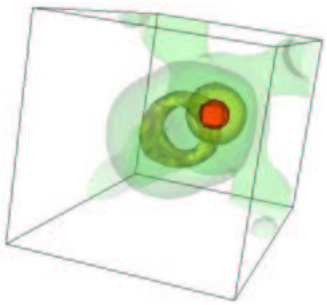


b: Random subsampled dataset, preventing aliasing.

Figure A.2: Aliasing effect during subsampling (top-view).

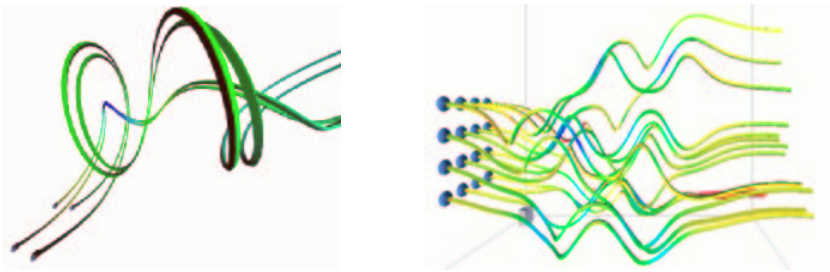


a: Low, mean and high value iso-surface



b: Top-perspective view.

Figure A.3: Iso-surface visualization of the vector-magnitudes (speed) of a whirlwind simulation.



Whirlwind. Mixer.

Figure A.4: Streamline examples.

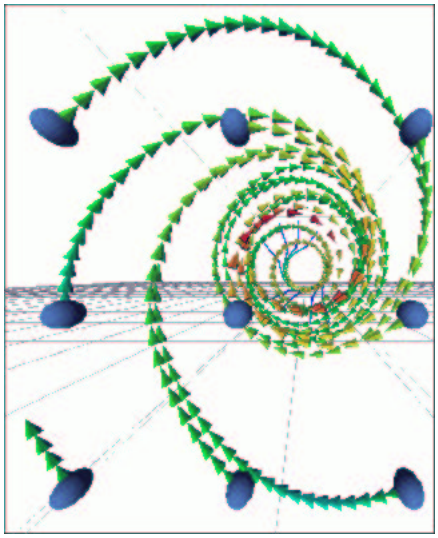
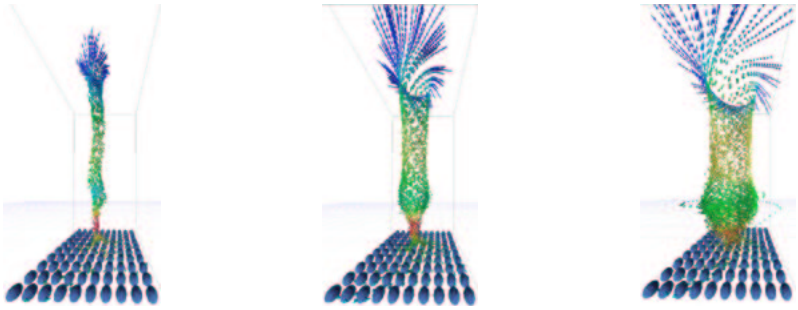


Figure A.5: Static particle example of a whirlwind simulation (bottom-up).



integration step 0.8 *integration step 1.6* *integration step 3.2*

Figure A.6: Impact of integration steplength.

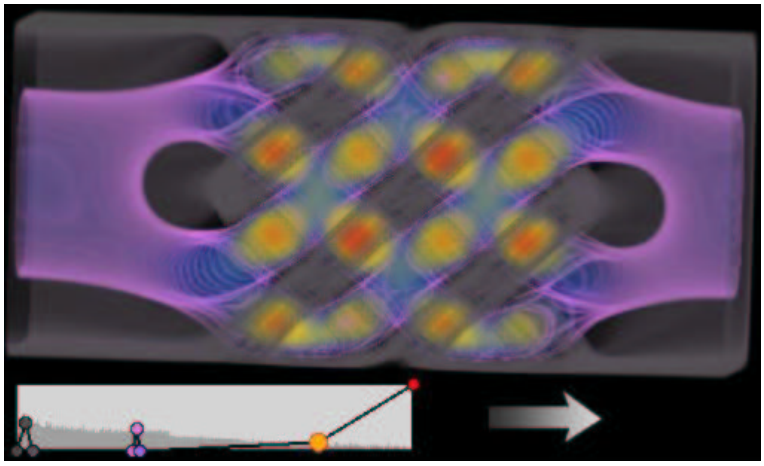
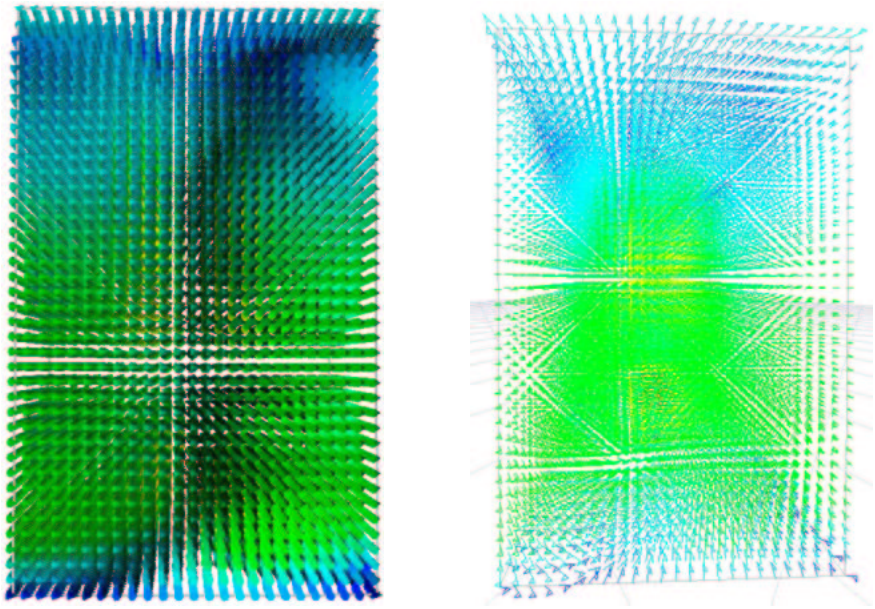


Figure A.7: Raycasted image of a 3D static reactor (SMRX) filter (see section 3.7 for a description) [48]. The arrow denotes the flow direction.



a: Cone visualization. *b:* Hook visualization.

Figure A.8: Glyph visualization of a whirlwind simulation without subsampling.

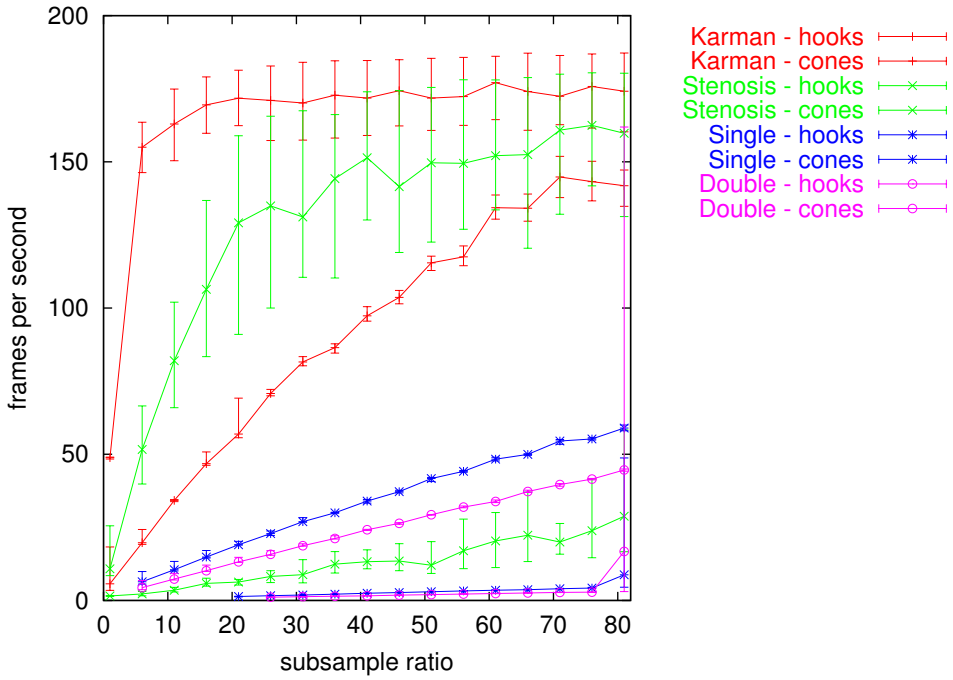


Figure A.9: Glyph visualization of several datasets, not synchronized with the vertical screen retrace.

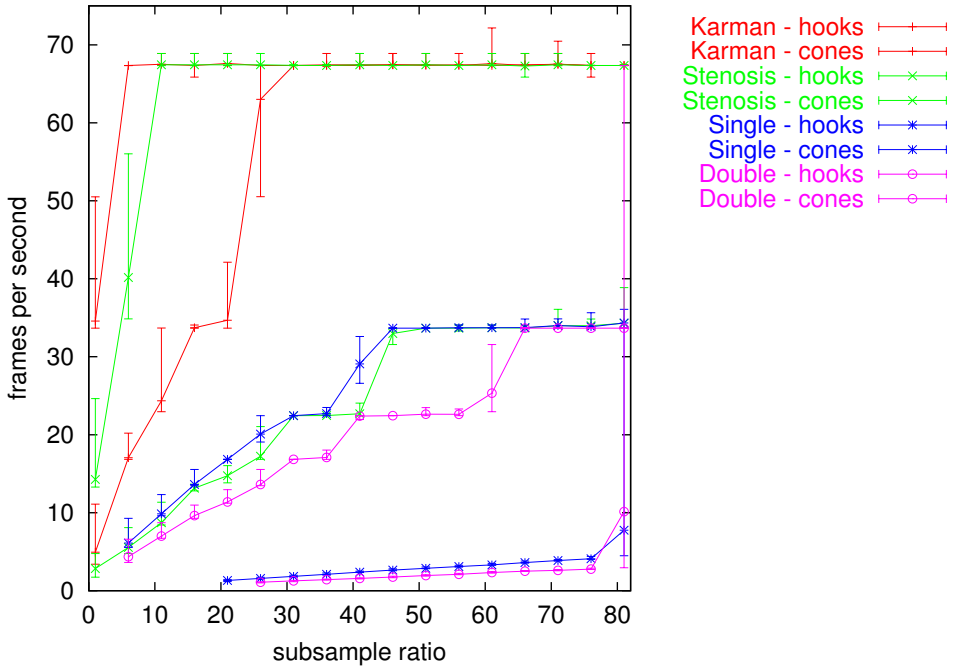
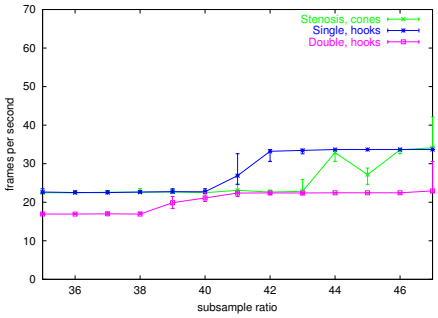
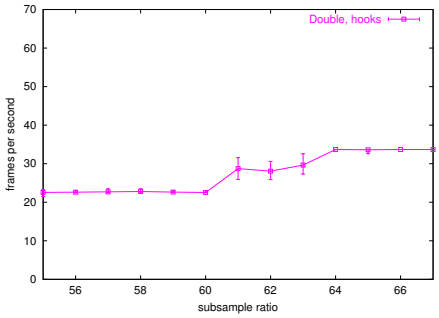


Figure A.10: Glyph visualization of several datasets, synchronized with vertical screen retrace.

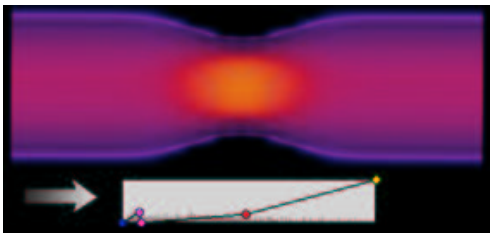


Stenosis, single and double mixer.

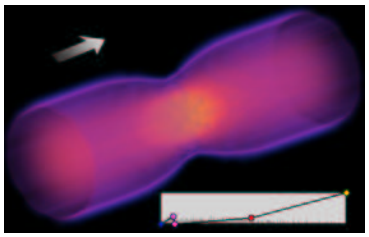


Double mixer.

Figure A.11: Hook and cone visualizations on subdomains using a step size of 1.

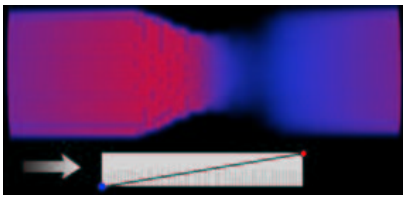


a: Front view

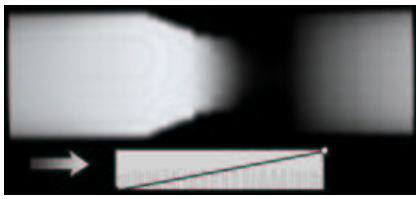


b: Perspective view.

Figure A.12: Stenosis speed visualization. The arrow denotes the flow direction.



a: Blue-red lookup table.



b: Black-white lookup table.

Figure A.13: Stenosis pressure visualization. The arrow denotes the flow direction.

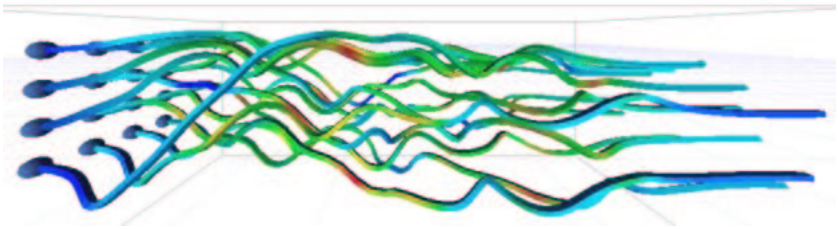


Figure A.14: Converging streamlines in double mixer.

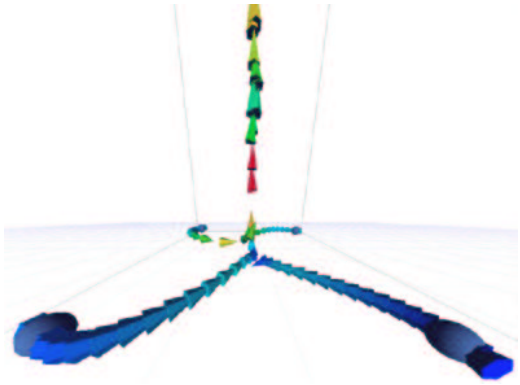


Figure A.15: Close-up of particles entering the nexus of the whirlwind.

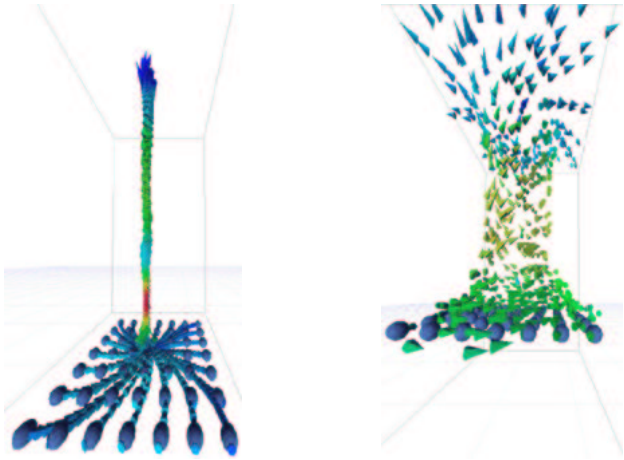
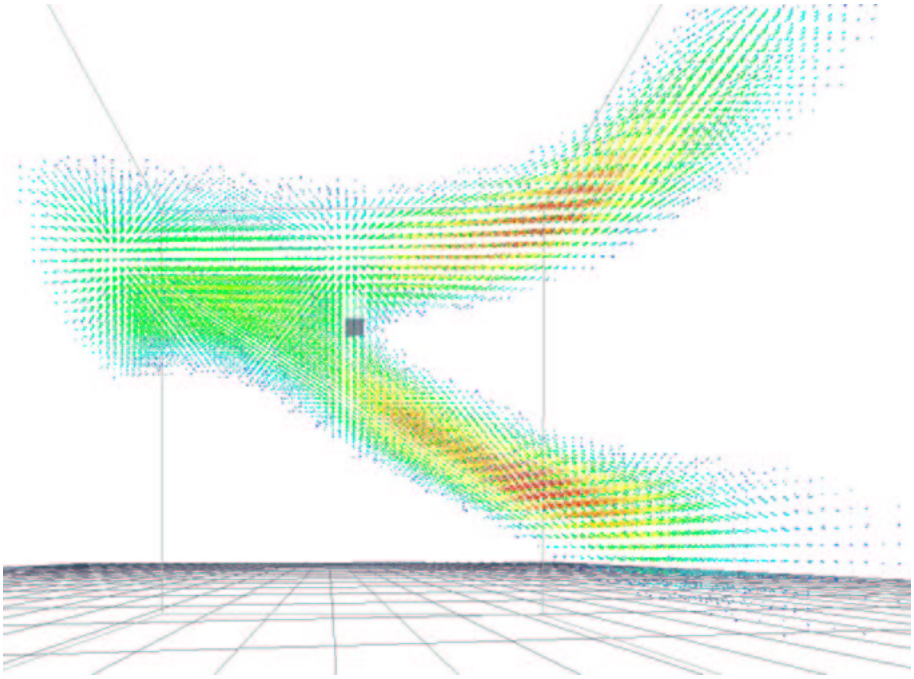
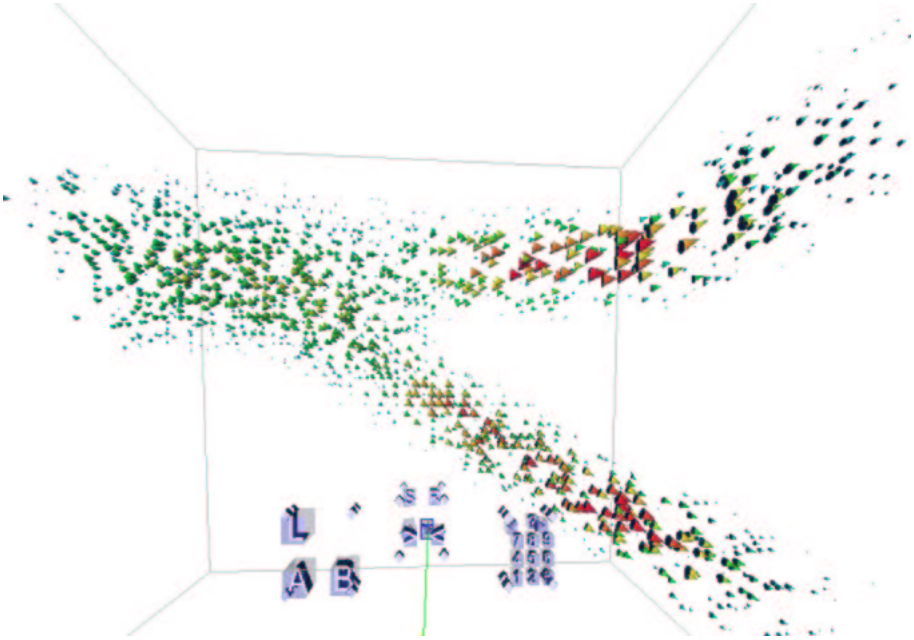


Figure A.16: Particles entering the whirlwind at different heights.

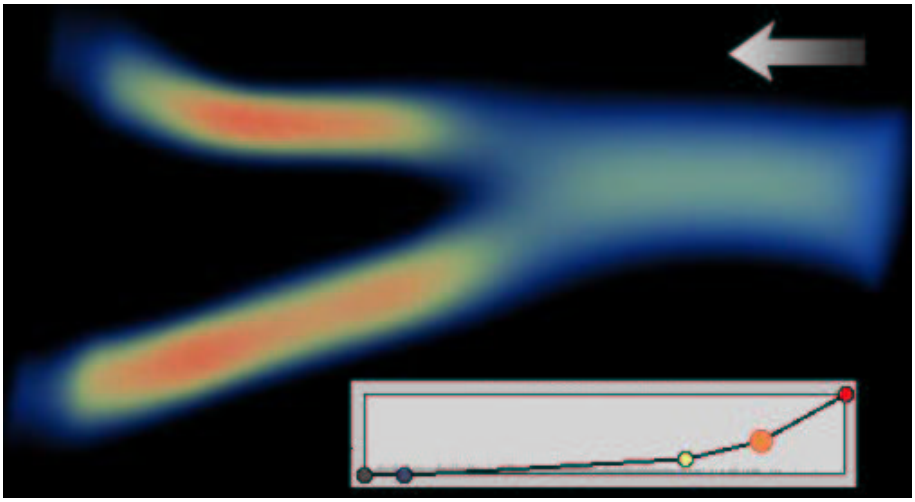


a: Hooks and no subsample ratio.

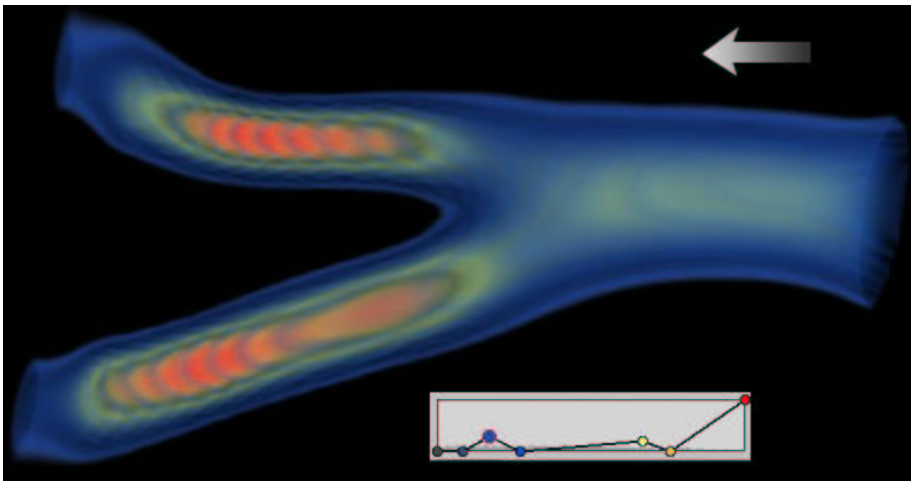


b: Cones and random subsampling.
Underneath, from left to right, flowfield menu,
glyphs menu and (glyphs-)subsample menu.

Figure A.17: Abdominal aorta.



a: Standard mapping.



b: Highlighted mapping.

Figure A.18: Front view of flow velocity in abdominal aorta.

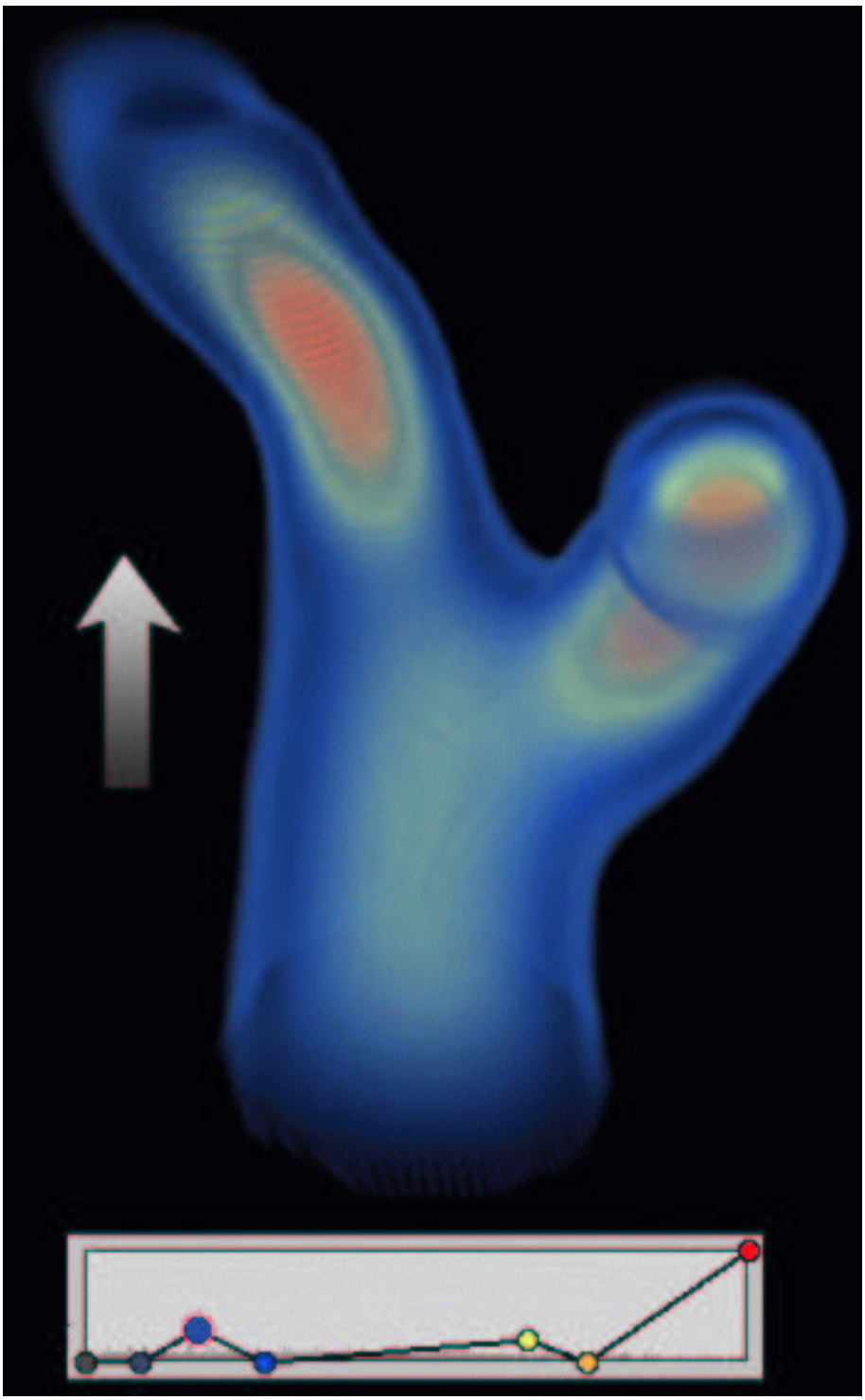


Figure A.19: Abdominal aorta (perspective view: top-front).

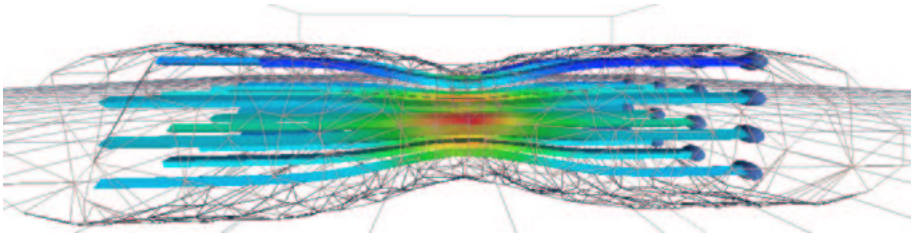


Figure A.20: Streamlines, originating from a plane cursor on the right, integrating through a stenosis.

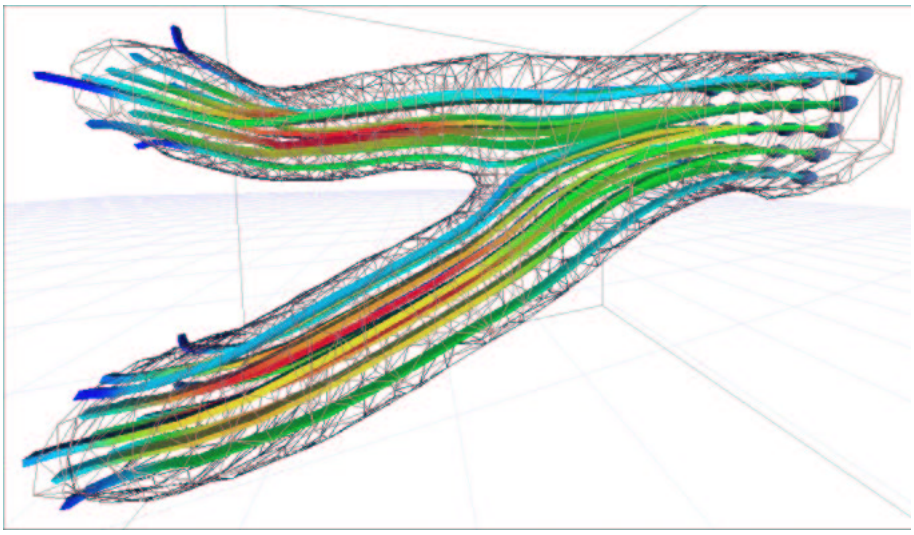
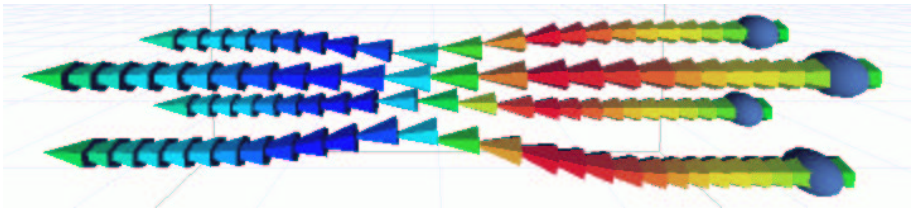
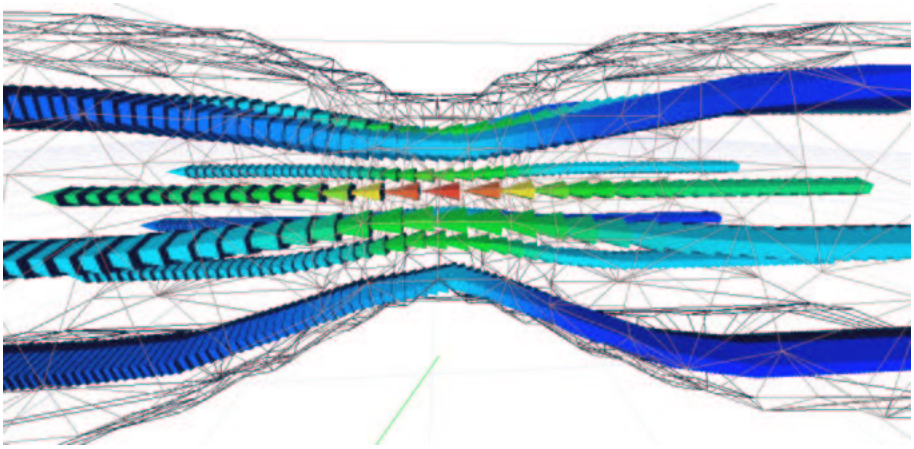


Figure A.21: Perspective view of aorta.



a: Coloured by pressure.



b: Coloured by speed, with artery wall representation.

Figure A.22: Close-up of particles through a stenosis.

Bibliography

[1] R.G. Belleman, Z. Zhao, G.D. van Albada, and P.M.A. Sloot. Design considerations for the construction of immersive dynamic exploration environments. In L.J. van Vliet, J.W.J. Heijnsdijk, T. Kielmann, and P.M.W. Knijnenburg, editors, *Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging*, pages 195–201, Lommel, Belgium, June 14-16 2000. Advanced School for Computing and Imaging (ASCI).

[2] R.G. Belleman, J.A. Kaandorp, and P.M.A. Sloot. Interactive environments for the exploration of large data sets. In B.M. ter Haar Romeny, D.H.J. Epema, J.F.M. Tonino, and A.A. Wolters, editors, *Proceedings of the fourth annual conference of the Advanced School for Computing and Imaging*, pages 264–268, Lommel, Belgium, June 9-11 1998. Advanced School for Computing and Imaging (ASCI).

[3] A. Schoneveld. *Parallel Complex Systems Simulation*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1999. Promotor: Prof. Dr. P.M.A. Sloot.

[4] G.A. Kessler, L.F. Hodges, and N. Walker. Evaluation of the cyberglove whole-hand input device. *ACM Transactions on Computer-Human Interaction*, 2(4):263–283, 1995.

[5] M. van Muiswinkel, L.H. Trip, and P.M.A. Sloot. Parallel sound synthesis: Parallel sonification of multivariate data on a transputer platform. In *Parallel Computing and Transputer Applications (PACTA 92)*, pages 1156–1164. University of Amsterdam, IOS Press, September 1992.

[6] E.R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, PO Box 430, Cheshire, CT 06410, 1999.

[7] OpenGL Organisation. OpenGL homepage, 2001. On the web: `http://www.opengl.org/`.

[8] A. v. Dam and J.D. Foley. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, June 1982.

[9] S. Bryson. Virtual environments in scientific visualization. Technical report, NASA Ames Research Center, Moffett Field and CA, 1995.

[10] Stan Posey, Mark Kremenetsky, and Ayad Jassim. Going with the flow, 1999. On the web: `http://www.memagazine.org/backissues/march99/features/theflow/theflow.html`.

[11] Shell and tube heat exchangers, 2000. On the web: `http://www.hyprotech.com/htfs/`.

[12] Sintef. Sintef homepage. On the web: `http://www.sintef.no/`.

[13] Simulation raises separator flow rate, March 2001. On the web: `http://www.engineeringtalk.com/news/flg/flg100.html`.

[14] A.M. Artoli, A.G. Hoekstra, and P.M.A. Sloot. Simulation of a systolic cycle in a realistic model of a human artery with the Lattice Boltzmann BGK method. In *11th internationl conference of discrete simulation of fluid dynamics and soft condensed matter*, August 2002.

[15] V.N. Vatsa and T. R. Faulkner. Navier-Stokes computations on commodity computers. Technical report, MRJ Technology Solutions and NASA Langley Research Center, Delphi, India, December 1998. 25th National and First International Conference on Fluid Mechanics and Fluid Power.

[16] W.A. Wood, C.J. Rilye, and F. McNeil Cheatwood. Reentry-f flowfield solutions at 80,000 ft. Technical Report NASA TM-112856, NASA Langley Research Center, Hampton and VA, May 1997.

[17] R.K. Bunner, J.C. Phillips, and L.V. Kale. Scalable molecular dynamics for large biomolecular systems. In *SC 2000: High Performance Networking and Computing Conference*. University of Illinois at Urbana-Champaign, 2000. Technical report 271.

[18] C. J. Reddy, Manohar D. Deshpande, C.R. Cockrell, and Fred B. Beck. Finite element method for eigenvalue problems in electromagnetics. Technical Report 3485, ViGYAN Inc and and Langley Research Center, December 1994.

[19] J.R. Hooker, A.W. Burner, and R. Valla. Static aeroelastic analysis of transonic wind tunnel models using finite element methods. In *AIAA Paper*, volume 2243. McDonnell Douglas Corporation and NASA Langley Research Center, American Institute of Aeronautics and Astronautics and Inc., June 1997. Presented at the 15th Applied Aerodynamics Conference.

[20] U. Frish, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier-Stokes equation. *Physical Review Letters*, 56:1505–1507, April 1986.

[21] D. Kandhai, D. Dubbeldam, A.G. Hoekstra, and P.M.A. Sloot. Parallel Lattice-Boltzmann simulation of fluid flow in centrifugal elutriation chambers. In *Lecture Notes in Computer Science*, volume 1401, pages 173–182, Kruislaan 403 and 1098 SJ Amsterdam, 1998. University of Amsterdam, Springer-Verlag. ISBN 3-540-64443-1.

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

[23] I. Trotts, D. Kenwright, and R. Haimes. Critical points at infinity: a missing link in vector field topology.

[24] J. van Wijk. Spot noise texture synthesis for data visualization. In *Computer Graphics 25*, July 1991.

[25] B. Cabral and L.C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH 93*. Computer Graphics Proceedings and Annual Conference Series, 1993.

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

[27] Numerical Algorithms Group. Irix explorer, 2002. On the web: `http://www.nag.co.uk/Welcome_IEC.html`.

[28] Silicon Graphics Inc. Silicon graphics homepage, 2002. On the web: `http://www.sgi.com/`.

[29] Advanced Visual Systems. Avs/express homepage, 2002. On the web: `http://www.avs.com/software/soft_t/avsxps.html`.

[30] IBM. OpenDX homepage, 2002. On the web: `http://www.opendx.org/`.

[31] Kitware Inc. Visualization toolkit homepage, 2001. On the web: `http://www.vtk.org`.

[32] R.G. Belleman, J.A. Kaandorp, D. Dijkman, and P.M.A. Sloot. GEOPROVE: Geometric probes for virtual environments. In P.M.A. Sloot, M. Bubak, A. Hoekstra, and L.O. Hertzberger, editors, *High Performance Computing and Networking (HPCN'99)*, pages 817–827, Amsterdam, the Netherlands, April 1999. Springer-Verlag.

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

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

[35] SARA Computing and Networking Services, Amsterdam and the Netherlands. *SARA Homepage*, 2001. `http://www.sara.nl/`.

[36] Polhemus. Fastrak, 2002. On the web: `http://www.polhemus.com/ftrakds.htm`.

[37] S. Bryson. Approaches to the succesful design and implementation of VR applications. In R.A. Earnshaw, J.A. Vince, , and H. Jones, editors, *Virtual Reality Applications*, pages 3–15. Academic Press, 1995.

[38] R.G. Belleman and P.M.A. Sloot. The design of dynamic exploration environments for computational steering simulations. In Marian Bubak, Jacek Mościński, and Marian Noga, editors, *Proceedings of the SGI Users' Conference*, pages 57–74, Kraków, Poland, October 2000. Academic Computer Centre CYFRONET AGH.

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

[40] R.G. Belleman and R. Shulakov. High performance distributed simulation for interactive simulated vascular reconstruction. In P.M.A. Sloot, C.J. Kenneth Tan, Jack J. Dongarra, and Alfons G. Hoekstra, editors, *International Conference on Computational Science (ICCS), Lecture Notes in Computer Science (LNCS) volume 2331*, pages 265–274, Amsterdam, the Netherlands, April 2002. Springer-Verlag, Berlin.

[41] R.G. Belleman and P.M.A. Sloot. Dynamic exploration environments. In Jeroen Meij, editor, *Dealing with the data flood (mining data, text and multimedia) (STT 65)*, pages 771–787. STT/Beweton, The Hague, the Netherlands, 2002.

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

[43] D. Hannema. Interaction in virtual reality. Master's thesis, University of Amsterdam and Section Computational Science, July 18 2001.

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

[45] Virtual Reality Consulting Inc., Chicago and IL. *CAVE User's Guide*, 1998.

[46] M. Hall. VTK to CAVE translator library, October 18 1999. On the web: `http://zeus.ncsa.uiuc.edu/~mahall/`.

[47] R.G. Belleman. Flexible interaction toolkits for immersive virtual environments. (to be pusblished).

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

[49] H.C.K. Sung. The area sampling machine. Master's thesis, University of Illinois at Urbana-Champaign and department of Computer Science, 1992.

[50] Silicon Graphics Inc. Software Products. Volumizer homepage, 2001. On the web: `http://www.sgi.com/software/volumizer/`.

[51] Kitware. Kitware's VolView homepage. On the web: `http://www.kitware.com/products/volview.html`.

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

[53] E. Hairer, S.P. Norsett, and G. Wanner. *Solving ordinary differential equations I: Non-stiff problems.* Springer-Verlag, Berlin, 1987.

[54] E.F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics and a practical introduction.* Springer-Verlag, Berlin, 1997.

[55] R.G. Belleman and P.M.A. Sloot. Simulated vascular reconstruction in a virtual operating theatre. In *CARS 2001 Conference*, Berlin and Germany, June 2001. University of Amsterdam.

[56] Screening for abdominal aortic aneurysm, April 2002. On the web: `http://cpmcnet.columbia.edu/texts/gcps/gcps0016.html`.

[57] L. Beolchi et al. *Telemedicine Glossary (3rd Edition).* DG Information Society Technologies, April 2001.

[58] Screening for asymptotic artery stenosis, April 2002. On the web: `http://cpmcnet.columbia.edu/texts/gcps/gcps0014.html`.

[59] A.M. Artoli, A.G. Hoekstra, and P.M.A. Sloot. 3D pulsatile flow with the Lattice Boltzmann BGK method. *IJMPC*, 8(13), October 2002.

[60] Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. System integration for interactive simulation systems using intelligent agents. In R.L. Lagendijk, J.W.J. Heijnsdijk, A.D. Pimentel, and M.H.F. Wilkinson, editors, *Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging*, pages 399–406, Heijen, the Netherlands, May 30-June 1 2001. Advanced School for Computing and Imaging (ASCI).

[61] J. van Wijk. Image based flow visualization. In *Proceedings of ACM SIGGRAPH 2002*, 2002. Computer Graphics Proceedings and Annual Conference Series.

All urls were checked and found valid on August 29^{th}*, 2002.*