7. International Immersive Projection Technologies Workshop
9. Eurographics Workshop on Virtual Environments (2003)
J. Deisinger, A. Kunz (Editors)

ViSTA FlowLib – A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments

M. Schirski[1†], A. Gerndt[1], T. van Reimersdahl[1], T. Kuhlen[1], P. Adomeit[2], O. Lang[2], S. Pischinger[3], and C. Bischof[1]

[1] Center for Computing and Communication, Aachen University (RWTH), Seffenter Weg 23, 52074 Aachen, Germany
[2] FEV Motorentechnik GmbH, Neuenhofstraße 181, 52078 Aachen, Germany
[3] Institute for Internal Combustion Engines Aachen (VKA), Aachen University (RWTH), Schinkelstraße 8, 52062 Aachen, Germany

Abstract

In the past a lot of work has been invested in various aspects of an interactive visualization of CFD simulation data. This includes e.g. increasing the rendering speed and responsiveness of complex visualizations, using and enhancing multimodal user interfaces, and incorporating parallel approaches for an efficient extraction of flow properties and their respective visual representation. Still, only few projects combine the significant advances in these areas. In this paper, we describe our software framework ViSTA FlowLib*, which facilitates merging current research results of various related areas. This is done by connecting dedicated sub-modules with clearly defined responsibilities through appropriate interfaces, whilst implementing sensible default behavior.* ViSTA FlowLib *combines efficient rendering techniques and a parallel computation of the visualization with intuitive multimodal user interfaces to allow for an interactive exploration of unsteady fluid flows in a virtual environment. Special care has been taken to achieve a high scalability in respect to computing power, projection technology, and input-output device availability.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Virtual Reality; I.3.6 [Computer Graphics]: Interaction Techniques; I.3.3 [Computer Graphics]: Viewing Algorithms; I.6.6 [Simulation and Modeling]: Simulation Output Analysis; H.5.2 [Information Interfaces and Presentation]: Graphical User Interfaces (GUI), User Interface Management Systems (UIMS); J.2 [Physical Sciences and Engineering]: Engineering

1. Introduction

A rapid increase in available computing power and significant improvements of simulation models have resulted in an ever increasing amount and complexity of simulation data being generated day by day. This has led to a need for sophisticated visualization methods to make optimum use of the results of these simulation runs. Especially the multi-dimensional nature of the data being generated complicates an intuitive understanding with standard two-dimensional visualization methods. In our opinion a combination of Virtual Reality (VR) methods and high-performance computing makes for an optimum approach to achieve an intuitive comprehension of the processes being simulated and visualized. Immersive display technology, stereoscopic and user-centered projection, and multimodal user interfaces ease the understanding of complex data, while a parallelized computation of the visualization improves its usability due to minimized response times.

Although there have been significant advances in the field of Virtual Reality, interactive flow visualization, and human computer interfaces in the last decade, only a few groups combined the respective novelties. We try to facilitate an incorporation of the corresponding advances into a single system by introducing our software framework *ViSTA FlowLib* for an interactive visualization and exploration of flow simu-

† Corresponding author, e-mail: schirski@rz.rwth-aachen.de

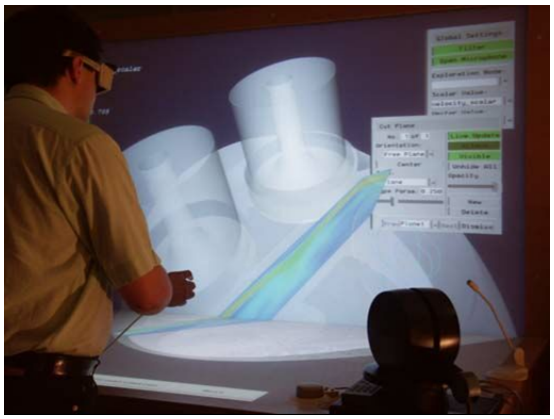

Figure 1: *Interactive exploration of a CFD dataset in a virtual environment. CFD data courtesy of: Aerodynamisches Institut, Aachen University (RWTH), Germany*

lation data in virtual environments. It is an extension library of our cross-platform virtual reality toolkit *ViSTA*[22] and uses the open-source Visualization Toolkit (VTK)[23] for a variety of visualization algorithms.

ViSTA FlowLib incorporates both well-known and innovative solutions to classical problems of an interactive visualization of Computational Fluid Dynamics (CFD) data in one flexible framework. Especially large-scale data handling, intuitive interaction with 4-dimensional data in virtual environments, and maximizing rendering speed are addressed. Additional features of our framework include a high scalability and strong extensibility.

We achieve this goal by decomposing the task at hand into three sub-modules with clearly defined interfaces and responsibilities, i.e. the multimodal user interface, a data stub for transparent communication with HPC work hosts, and a central part for rendering and time management. First prototypes, which allow an interactive visualization of path lines in an unsteady CFD dataset of an internal combustion engine, show considerable improvements with regard to needed computation time, visualization control, and rendering speed. In this case, parameters and seed points for the pathlines are chosen via an intuitive multimodal interface, and the trajectories of the corresponding massless particles are computed on high-performance SMP clusters and rendered efficiently to immersive display systems by a dedicated visualization host, resulting in an intuitive and interactive visualization of the simulated fluid flow. Every component of this system is scalable, which allows for using lower-end hardware as well, i.e. commodity PCs for rendering purposes or LINUX clusters for tracing the particles. It is even possible to use a single workstation for the whole process, if the user accepts considerable performance losses with regard to computation speed and responsiveness of the system.

Future extensibility is maintained by keeping the single components as independent as possible. This allows for the incorporation of e.g. different parallelization or visualization approaches by just substituting the respective part of the system. Especially the combination with VTK allows for a transparent inclusion of new visualization algorithms, which are constantly being developed and implemented by the corresponding open-source community.

Note, that our goal is not only the implementation of another visualization toolkit, but an open library, which facilitates the implementation of CFD flow visualizations by offering a solid amount of visualization and parallelization capabilities, paired with intuitive user interface options.

The remainder of this paper is structured as follows: The next section gives an overview of related work, followed by a description of the fundamental building blocks of the framework. The sections 4 to 6 discuss these building blocks in more detail, before a prototypical implementation, which makes use of *ViSTA FlowLib*, is described. The last section consists of some conclusions and an outlook for future work.

2. Related Work

Visualizing complex fluid flows in virtual environments is a goal, which has been pursued by a diversity of research groups with various objectives and approaches. Significant progress has been achieved in single or few aspects of the visualization process. In the remainder of this section, we give examples for successful work in various related areas.

Concerning human computer interfaces, Laviola[16] and Ebert and Shaw[7] concentrate on two-handed user-interfaces, but seem to rely on an in-core computation of the visualization, which limits the amount of data that can be processed.

An intuitive visualization of fluid flows is achieved through a variety of methods[19]. As we consider integral objects like streamlines or particle traces to be especially intuitive, we concentrate on examples for using those as given e.g. by Lane[14]. Zöckler et al[25] and Fuhrmann and Gröller[8] describe methods for an improved perception and basic user interfaces in virtual environments for the visualization of streamlines. An out-of-core pre-computation and sophisticated encoding of particle traces is implemented by Bruckschen et al[2], along with an interface for choosing which particle traces to display at any given time.

The exploration of larger datasets is sped up by incorporating a parallelized computation of time consuming extraction algorithms on dedicated work hosts, e.g. with focus on parallel computation[15] or load balancing[1]. Streaming of (parallel) pre-computed visualization data to a visualization host was considered by Olbrich et al[18].

Whereas most of the given examples concentrate only on a few aspects, some systems combine several important approaches to achieve powerful visualization schemes. This

includes COVISE and its VR add-on COVER[20], which allows sophisticated visualization features to be computed on parallel HPC systems and displayed in virtual environments. Data transfer efficiency between the work host and the visualization host is improved by data compression strategies[12]. However, the VR user interface consists mainly of a pointing device and basic 3D menus.

Other examples for dedicated HPC work hosts being used for the computation of a fluid dynamics visualization to relieve the visualization host are the Distributed Virtual Windtunnel[4], which is an extension of Bryson's and Levit's original Virtual Windtunnel[3], and the Responsive Workbench[13, 24], which both focus on a certain type of display system and interaction metaphor.

Work, which is conceptually related to ours, has been done by Bryson with a framework for the Virtual Windtunnel[5]. It offers the possibility to add new visualization and interaction methods as well, but is limited to SGI platforms and does not address the data acquisition part of the visualization pipeline.

3. Fundamentals

ViSTA FlowLib uses our cross-platform VR-toolkit *ViSTA*[22] for the implementation and management of sophisticated VR functionality like access to immersive display systems, motion trackers, and haptic output devices (see figure 2). As *ViSTA* is highly scalable in respect to available hardware, this allows for a seamless integration of the required functionality on a diversity of computing platforms, ranging from standard commodity PCs to high-end HPC systems and from common computer monitors to multi-screen immersive displays. *ViSTA* relies on a dedicated scenegraph API for scenegraph management. For now, it uses the commercial WorldToolKit (WTK) from Sense8[21], but we are planning to replace it with an open-source toolkit.

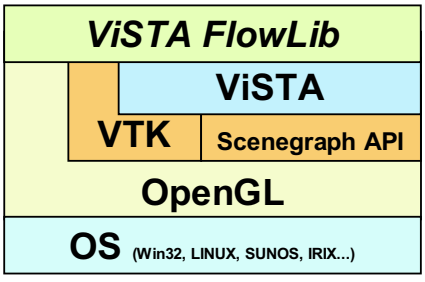

Figure 2: ViSTA FlowLib *is an extension to* ViSTA *(Virtual Reality Toolkit University of Technology Aachen), which in turn is based on* VTK *and a scenegraph API.*

In addition, we build upon the open-source Visualization Toolkit (VTK)[23] for an implementation of several visualization algorithms like isosurfaces and cut planes. As VTK does not include functionality for time-dependent visualization (yet), we substitute the required functionality as needed. Examples include data management and pathline extraction algorithms. Furthermore, we use VTK data structures for a variety of communication processes between the parallelized flow property extraction and the visualization part of the framework. In some cases we utilize a part of the VTK rendering pipeline for drawing standard geometry as well, by using appropriate methods of single visualization objects, which in turn issue OpenGL commands to the graphics system.

As depicted in figure 3, our framework *ViSTA FlowLib* itself is comprised of several parts, which include a central rendering and time data management part, the multimodal user interface with a variety of default functionalities, and a data stub for communication with dedicated HPC work hosts.

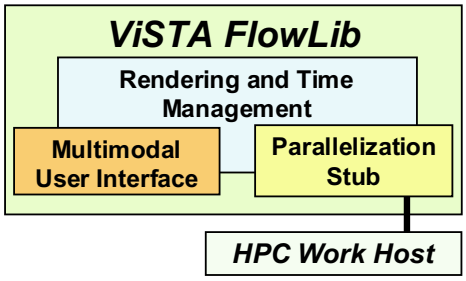

Figure 3: ViSTA FlowLib *includes a central data management part, the multimodal user interface, and a data stub for communication with dedicated HPC work hosts.*

4. Time Management and Rendering

Within the time management and rendering part of the *ViSTA FlowLib* framework timing information for the unsteady datasets is provided. In addition, access to graphical resources is controlled. To achieve a maximum flexibility, responsibilities are distributed according to functional entities, each one being as independent as possible of the rest of the system.

Time management and rendering are implemented by the *visualization controller*, several *visualization objects*, (potentially multiple) *time mappers*, and *resource managers* (see figure 4). In the following sections these objects are briefly described, before further details about the human computer interface and parallelization capabilities are presented.

4.1. Visualization Controller

The *visualization controller* is the central entity of the visualization process. It manages all visualization objects by offering interfaces for run-time registration and unregistration,

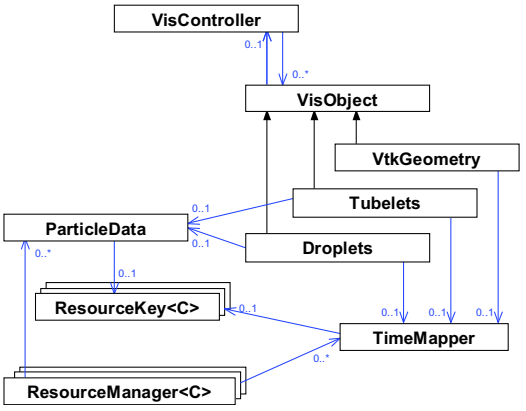

Figure 4: *In the time management and rendering part of the framework responsibilities are distributed according to functional entities. Besides the central visualization controller (`VisController`) examples for implementations of visualization objects (`Droplets`, `Tubelets`, and `VtkGeometry`) and shared resources (`TimeMapper` and `ParticleData`) are depicted, along with the respective resource key and resource manager objects.*

as well as run-time data queries. In addition, it synchronizes all visualization objects by managing visualization time and calling their respective update and rendering methods.

4.2. Visualization Objects

Visualization objects represent any kind of entity, which has to have access to graphical resources of the visualization host, i.e. anything that is to be rendered on screen or other display devices. As these objects have direct access to the graphics hardware of the visualization host, they can use highly optimized rendering algorithms and e.g. exploit special features of modern graphics systems. This allows for maximum rendering performance and even permits programmable graphics hardware to be used.

As a render loop is split up into several "passes", including the drawing of opaque objects, the drawing of transparent objects, and drawing 2D elements, visualization objects register with the visualization controller for the respective callbacks. When an objects is called within the render loop, it has the opportunity to query the visualization master for runtime information (e.g. the current visualization time or the position of the viewer) before it renders itself via OpenGL commands. Independent of the source of the visualization data, i.e. a file or a parallelized computation, the visualization object holds all data, which is needed for the graphical representation, and selects the data to be drawn according to the current visualization time level. If possible, all CFD data is kept and managed on the HPC work host, and only visualization data is transferred to the visualization host.

The benefits of this approach include high flexibility and extensibility of the system. As every visualization object is independent of any other object, new visualization algorithms can be incorporated into the framework at will.

4.3. Resource Management

To avoid multiple instances of equivalent resources (e.g. datasets, time mappers etc.), templatized *resource managers* are introduced. In combination with appropriate *resource keys*, which uniquely identify specific resources, access to resources is provided. If a resource is requested, the resource manager decides with the help of the given resource key, if the resource exists already. If it does, a reference to it is returned; if it does not exist, it is created according to the information given in the resource key, before the respective reference is returned. Thus, a resource key must contain all information that is necessary for creating the requested resource. Reference counting enables a resource manager to destroy resources, if they are not in use anymore. This way, a minimum amount of resources is allocated for the current visualization, because data is shared wherever possible and resources are freed as soon as possible.

Typical examples for shared resources are pre-computed data like graphical representations of the bounds of the flow domain or the behavior of liquid fuel droplets within the simulated fluid flow, which belong to the resulting data of the numerical simulation. In these cases the file names are part of the respective resource keys. Of course, a particle data object, which acquires its data through the parallelization stub, can act as a shared resource as well. In this case, its data can be shared to be displayed by two different visualization objects, which implement different visualization methods, e.g. path lines and spherical particles.

A typical life cycle of a visualization object is as follows: After it is created, it registers with the visualization controller to get access to the graphics system, before it requests necessary resources from the resource manager. During its lifetime its update method is called to allow the object to update itself according to its resources and the state of the visualization, which can be queried e.g. from the visualization controller. Subsequently, its render methods are called to display itself. When the visualization is finished, the object is destroyed and all resources are freed by the resource manager, if they are not used by another object.

4.4. Time Mapping

When dealing with unsteady flow data, different visualization objects and their corresponding data potentially exist in different time frames, especially if they are integral objects or results of different simulation runs. To facilitate the handling of these cases, a flexible time mapping via dedicated *time mappers* is incorporated into the framework. They are responsible for mapping normalized visualization time, i.e.

time values within $[0, 1]$, to absolute simulation time and the corresponding simulation time step or vice versa. This allows for an accurate depiction of integral objects like pathlines, whose integration steps do not necessarily coincide with the simulation time steps. Furthermore, it facilitates a comparison of simulations that are computed within different time frames. In addition, the results of several runs, which simulate different events of a process, can easily be assembled into a single, comprehensive visualization.

5. Multimodal Human Computer Interface

The human computer interface of our framework *ViSTA FlowLib* facilitates the application development. The framework provides a default human computer interface, which can be configured individually. Besides the graphical user interface, the multimodality of the human computer interface concerns also speech input, audio output, and haptic rendering methods of the visualization algorithms. Each visualization technique possess a default user interface with which the visualization parameters can be modified interactively. In addition, the application developer is able to build customized user interfaces. In this case, he has to implement the appropriate abstract interfaces to fulfill his requirements.

The framework of the Multimodal Human Computer Interface is divided into one sub-framework for user interfaces and one sub-framework for a Multimodal Input Output Manager (MIOM).

The basic idea of the menu user interface framework consists of a set of abstract menu classes, which can be implemented as needed. Thus, an application can use different types of menus like geometry-based 3D menus (Fig. 1), texture-based 3D menus, and remote user interface menus for Tablet PC menus and speech input. At run-time, the users can switch the menu type depending on their current preference.

In the default user interfaces, we have currently implemented three levels of complexity. The lowest one offers texture-based 3D menus, which are usually attached directly to a single input device such as a tracked wand. Here, only the most important menu elements are accessible. The middle level implements geometry-based 3D menus, where the user can modify more visualization parameters. The most complex level provides a remote 2D Java GUI on a wireless Tablet PC. To avoid user confusion after switching to another menu type, we kept the menu hierarchy elements and the parameters of all menu elements in the next higher level and extended the access to more parameters.

To maintain consistency between the related menu types and the visualization parameters we use the Observer pattern[9]. The subject class of this pattern is the parameter class referenced by the visualization object class. The observer classes are the different menu type classes. Whenever the parameters of a visualization object (the subject) changes its state, all observers are notified. In response, the observers synchronize theirs state with the current state of the subject.

In the MIOM framework, input and output events from several modal interfaces are controlled. Here, events awaiting processing are analyzed and transformed into other events according to default and/or user-defined rules. For instance, if the user wants to play a sound after a visualization event is completed, he simply needs to define a rule which adds a new sound event to the visualization event type. Another example is the processing of tracked input devices. If the user wants to change the command assignment of the buttons of a tracked joystick, he only needs to change the appropriate rules (at run-time). In this fashion, this MIOM approach allows for a great flexibility in using input and output devices.

Figure 1 shows an example of the MIOM in action on a HoloBench projection system. A color-coded cut plane is interactively positioned with a tracked wand. Additionally, the user can rotate the cylinder geometry of a spark ignition engine via voice commands.

6. Parallelization

Higher frame-rates can be achieved by using special visualization techniques and optimized graphics hardware. For CFD post-processing, the objects to be displayed have to be computed first. This extraction computation is often time consuming, and the used data sets are generally huge. With highly integrated interaction facilities, computations for the post-processing are performed continuously. This is a big challenge for the CFD framework.

Because graphics workstation hardware is optimized for the rendering process and normally works at its limit in order to offer high frame-rates, CFD post-processing prevents real-time rendering and interaction. Multi-threading on multi-processor graphics workstations can reduce this problem, but for demanding post-processing this is not a satisfying solution.

Therefore, extraction computations and CFD data management are taken over by a dedicated high-performance computing (HPC) cluster. Our developed stub-object located at the visualization host receives computation requests, sends computation inquiries to the HPC work host, and manages pending computation tasks. Results received from the work host are converted to displayable objects for the rendering process. The communication between stub-code and rendering loop is carried out by multi-threading, and the communication between stub-code and the work host uses TCP/IP, but can fall back on MPI as well. The stub-code abstracts from implementation details; i.e., it hides the location of computation, which can be carried out either in parallel on a remote HPC work host or sequentially on the local workstation.

6.1. 3 Layers

The work host consists of a scheduler, which schedules the incoming requests, and a variable amount of workers for the parallel computation. These processing elements (PEs) communicate by messages. This communication is toolkit independent, although it corresponds to a message passing scheme[10].

In order to make our parallelization approach more flexible, we have defined three different communication layers. The middle one contains scheduler, worker, and additional support classes for communication between them. The lowest one contains the actually used transport channel between different processes. Here, one can select TCP/IP for the communication to the visualization host and MPI[11], the most common message passing toolkit right now, for the communication between different PEs on the work host. In some special situations, MPI or MPICH can also be used between visualization host and work host. This layer concept with clearly defined interfaces simplifies the implementation of other parallelization toolkits and communication protocols.

The highest layer implements the required post-processing algorithms. These algorithms are responsible for balancing and speed-up. Likewise, loading data from a file server is managed here. This is a very important aspect because loading data might be a crucial bottleneck. Right now, each PE handles its own data access. For higher performance, a more common data management is going to be developed. This algorithmic layer is not restricted to CFD post-processing but can also be exploited for all sorts of parallelizable computations.

6.2. ccNUMA

The described parallelization concept is based on message passing and therefore works on off-the-shelf systems like LINUX-clusters as well as on some million dollar high-performance computers[10]. Generally, expensive high-performance clusters make use of shared memory. Therefore, we are interested in optimizing the parallelization approach for shared memory systems as well. Our main high-performance system is a Sun-Fire cluster by Sun Microsystems with 672 CPUs, 906 GByte main memory and an accumulated peak performance of 1.2 TFlops/s, which is located at the Center for Computing and Communication of Aachen University.

This system makes use of an optimized MPI version, which uses shared memory in order to send messages and data, but cache coherency and ccNUMA aspects are not considered. Therefore, our parallelization framework is being developed into an extended version that additionally uses OpenMP[6], which exploits these advanced hardware features. However, in our concept, OpenMP will not compete with MPI but will be applied on the algorithmic layer as a supplementation. Specialized SMP algorithms should increase the already achieved speed-up even more.

6.3. Streaming

Our parallelization approach considerably reduces the time a user has to wait for computation results in a virtual environment. Furthermore, real-time interaction within this virtual environment is made possible. However, the acceptance for this implementation can suffer if the response time exceeds a certain limit.

In order to avoid this, a concept called streaming is going to be implemented. Partly calculated results are continuously sent to the visualization host and are visualized directly. The user receives a first impression of the upcoming final result almost immediately. On this occasion, one can already begin to evaluate the results. If the computation parameters seem to be sub-optimal, the computation request can be withdrawn and restarted with new parameters. Therefore, not only the acceptance can be increased, but also the assessment of the flow field can be sped up.

Right now, techniques to compute partial results, which can be sent back as a continuous stream, are investigated. We distinguish between calculations based on the Eulerian formulation and on the Lagrangian formulation of the flow field. A variation of multi-resolution approaches is published for extraction algorithms of the first category[18], e.g. iso-surfaces. Streamlines and pathlines, which are derived from the Lagrangian formulation, are more complicated. Multi-resolution approaches are combined with increased time consumption. However, pieces of calculated polylines can already be transmitted and displayed.

7. Prototypical Application

We implemented a prototype with *ViSTA FlowLib* to demonstrate the advantages of our framework in achieving a powerful fluid flow visualization. In the remainder of this section, we discuss an intuitive and interactive approach to pathline computation and visualization, which makes heavy use of our framework. Apart from free navigation within the fluid flow domain, it allows a user to intuitively control the creation of pathlines by modifying the seed points and other parameters via a multimodal user interface. Once the seed points are chosen, they are passed to remote HPC work hosts, which compute the particle trajectories and return this data to the visualization host. Finally, the particles and their trajectories are displayed in a virtual environment efficiently, along with other contextual data like the bounds of the flow domain.

The implemented user interface is highly scalable with respect to the working environment and available equipment, ranging from sophisticated immersive display systems with motion trackers to standard desktop computers with a mouse

and keyboard setup. If the appropriate hardware is available, the multimodal interface allows the use of 3D pointing devices, speech recognition, and haptic rendering. On a commodity PC the visualization can be controlled via mouse and keyboard as well.

As mentioned above, the particle trajectories are computed in parallel on dedicated HPC work hosts. Parallelization is done by distributing the given seed points to available work hosts, which compute the complete trajectory of the respective particle. This results in a considerable speed-up in comparison to a serial computation. The parallelization is done transparently for the visualization host, i.e. if there is no HPC cluster available, the particle tracks can be computed even on the visualization host itself, although a significant performance loss must be tolerated.

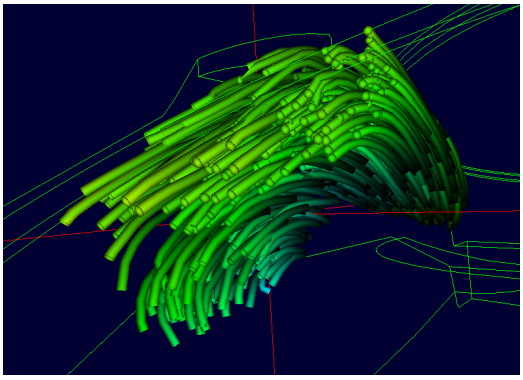

Figure 5: *Particle tracks are rendered as Virtual Tubelets consisting of viewer-aligned polygons, which fake the illumination of rounded geometry by appropriate textures.*

The rendering part of the prototype consists of several visualization objects. Apart from basic VTK geometry display objects, which use standard VTK rendering functionality, we implemented objects for an efficient depiction of particle data and pathlines of high visual quality by using and extending billboarding techniques[17]. To enhance the visualization of large amounts of particle traces we developed *Virtual Tubelets* – billboarded and capped stream tube sections, which create the illusion of slightly self-illuminated, rounded objects by faking illumination via appropriate texture maps (see figure 5). By rotating the billboards towards the viewer we maintain the illusion of volumetric objects even in virtual environments with multiple projection screens. Compared to conventional stream tubes, rendering speed is increased significantly due to much less polygonal complexity, while still maintaining good visual quality and even enhancing flexibility. To further enhance visual quality we use fake lighting via texture mapping on the billboarded droplets, too. A comparison of geometrical cubes and the billboarded particles is shown in figure 6, clearly showing that the billboarded approach is superior with regard to visual quality. The same applies to rendering speed.

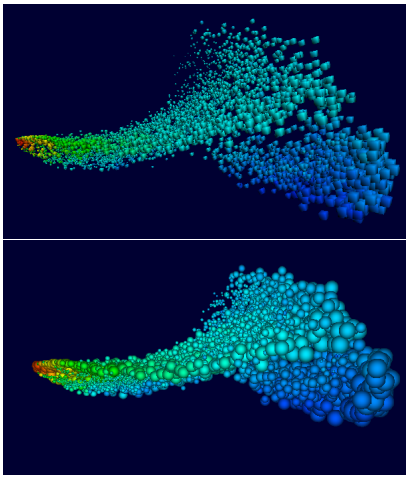

Figure 6: *The visual quality of a cuboid representation (top) of particles is far inferior to a billboarded representation (bottom).*

Once the particle trajectories are computed, the user interface can again be used to control further parameters of the visualization.

Figure 4 shows the main objects of the rendering part of the prototype. The `Droplets` and `Tubelets` are implemented as specializations of `VisObject`, data is shared through the `ParticleData` object, which in turn is identified by a `ResourceKey` and created by a `ResourceManager`. Depending on the contents of the resource key, particle data is loaded from a file or received from the parallelization stub. In the latter case, appropriate user interface elements for the control of the seeding points are created.

The development of the rendering methods, the parallel pathline computation, and the basic user interface modules have been done by three disjunct groups. The implementations have been added to the single library *ViSTA FlowLib*, which was later used to implement the prototype.

8. Conclusions and Future Work

We have defined a framework to combine efficient VR rendering techniques, scalable, multimodal user interfaces, and parallelization capabilities into a single powerful cross-platform CFD data exploration library, which facilitates the implementation of sophisticated flow visualizations considerably. Solutions and default behavior are provided for unsteady data and time management, parallelized flow property computation, rendering techniques, and intuitive and scalable user-interfaces. In addition, every single part of the library is extensible to account for future advances in any of the respective fields of flow visualization. First prototypes showed promising advances for a scalable, intuitive, and interactive flow visualization.

For the future, we have planned to further extend *ViSTA FlowLib* by making use of programmable graphics hardware to further improve on the quality and speed of the rendering of the visualization. This includes e.g. refining the lighting of the *Virtual Tubelets* through vertex and/or fragment programs. Using our framework for texture-based volume visualization will play an important role within our future work as well.

Regarding the parallelization part, we will implement additional flow property extraction algorithms. In addition, advanced LOD and streaming mechanisms are going to be incorporated to improve the visualization's responsiveness during the parallelized computation of the visualization.

With respect to the multimodal human computer interface part, we will refine and extend the interaction techniques, evaluate them, and investigate the contribution to a faster knowledge acquisition by performing user studies.

References

1. C.L. Bajaj, V. Pascucci, D. Thompson, X.Y. Zhang. "Parallel accelerated isocontouring for out-of-core visualization". *Proceedings of Parallel Visualization and Graphics Symposium*, San Francisco, California, 1999.
2. R. Bruckschen, F. Kuester, B. Hamann, K.I. Joy. "Real-time Out-of-core Visualization of Particle Traces". *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pp. 45–50, 2001.
3. S. Bryson, C. Levit. "The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows". *Proceedings of Visualization '91*, pp. 17–24, 1991.
4. S. Bryson, M. Gerald-Yamasaki. "The Distributed Virtual Windtunnel" *Proceedings of Supercomputing '92*, Minneapolis, Minnesota, November 1992.
5. S. Bryson, S. Johan, L. Schlecht. "An Extensible Interactive Visualization Framework for the Virtual Windtunnel". *Proceedings of the Virtual Reality International Symposium*, pp. 106–113, 1997.
6. R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon. *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, 2000.
7. D.S. Ebert, C.D. Shaw. "Minimally-Immersive Flow Visualization". *IEEE Transactions on Visualization and Computer Graphics*, **7**(4):343–350, 2001.
8. A. Fuhrmann, E. Gröller. "Real-Time Techniques For 3D Flow Visualization". *Proceedings of Visualization '98*, pp. 305–312, 1998.
9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.
10. A. Gerndt, T. van Reimersdahl, T. Kuhlen, C. Bischof. "A Parallel Approach for VR-based Visualization of CFD Data with PC Clusters". *Proceedings of IMACS 2000*, Lausanne, Switzerland, 2000.
11. W. Gropp, E. Lusk, A. Skjellum. *Using MPI - Portable Parallel Programming with the Message Passing Interface*, 2nd edition, MIT Press, 1999.
12. H. Jasak, J.Y. Lui, B. Kaluderčić, A.D. Gosman, H. Echtle, Z. Liang, F. Wirbeleit, M. Wierse, S. Rips, A. Werner, G. Fernström, A. Karlsson. "Rapid CFD Simulation of Interanl Combustion Engines". *SAE International Congress and Exposition '99*, 1999.
13. W. Krüger, C.-A. Bohn, B. Fröhlich, H. Schüth, W. Strauss, G. Wesche. "The Responsive Workbench: A Virtual Work Environment". *IEEE Computer*, **28**(7):42–48, 1995.
14. D.A. Lane. "UFAT – A Particle Tracer for Time-Dependent Flow Fields". *Proceedings of Visualization '94*, pp. 257–264, 1994.
15. D.A. Lane. "Parallelizing a Particle Tracer for Flow Visualization". *7th SIAM Conference on Parallel Processing for Scientific Visualization*, San Francisco, California, February, 1995.
16. J.J. Laviola Jr. "MSVT: A Virtual Reality-Based Multimodal Scientific Visualization Tool". *Proceedings of the Second IASTED International Conference on Computer Graphics and Imaging*, pp. 221–225, 1999.
17. T. Möller, E. Haines. *Real-Time Rendering*, A K Peters Ltd., 1999.
18. S. Olbrich, H. Pralle, S. Raasch. "Using Streaming and Parallelization Techniques for 3D Visualization in a High-Performance Computing and Networking Environment". *Proceedings of High-Performance Computing and Networking*, 2001.
19. F. Post, B. Vrolijk, H. Hauser, R.S. Laramee, H. Doleisch. "Feature Extraction and Visualization of Flow Fields". *State-of-the-Art Proceedings of Eurographics 2002 (EG 2002)*, pp. 69-100, 2002.
20. D. Rantzau, U. Lang. "A scalable virtual environment for large scale scientific data analysis". *Future Generation Computer Systems*, **14**(3-4):215–222, 1998.
21. `http://www.sense8.com`.
22. T. van Reimersdahl, T. Kuhlen, A. Gerndt, J. Henrichs, C. Bischof. "ViSTA: a multimodal, platform-independent VR-Toolkit based on WTK, VTK, and MPI". *Proceedings of Fourth International Immersive Projection Technology Workshop (IPT2000)*, Ames, Iowa, 2000.
23. W. Schroeder, K. Martin, and W.E. Lorensen. *The Vi-*

sualization Toolkit: An ObjectOriented Approach to 3D Graphics, 2nd edition, Prentice Hall, 1998

24. G. Wesche, J. Wind, M. Göbel. "The Responsive Workbench for Visualization of Fluid Dynamics". *ERCIM News*, **31**:37–39, October 1997.

25. M. Zöckler, D. Stalling, H. Hege. "Interactive Visualization Of 3D-Vector Fields Using Illuminated Stream Lines". *Proceedings of Visualization '96*, pp. 107–113, 1996.