Design and Implementation of Geometric and Texture-Based Flow Visualization Techniques

Robert S. Laramee, Markus Hadwiger, and Helwig Hauser*
VRVis Research Center
Donau-City-Strasse 1
Vienna, Austria
www.VRVis.at

Abstract

Usually, research related software consists of individual, isolated prototypes because researchers are interested in a small proof-of-concept application for demonstration. Here we present software developed for research purposes, but which has been included into a larger, commercial visualization system. We describe the design and implementation of a flow visualization subsystem within the framework of a software package capable of modeling, simulation, and visualization of CFD simulation data. Our flow visualization subsystem provides several research related geometric and texture-based visualization techniques. As a result, we are able to combine visualization options in new ways that typical research prototypes cannot. Here we describe some of our design and implementation decisions and outline the resulting advantages and disadvantages.

CR Categories: I.3.0 [Computer Graphics]: General I.3.2 [Computer Graphics]: Graphics Systems I.3.4 [Computer Graphics]: Graphics Utilities I.3.8 [Computer Graphics]: Applications I.3.m [Computer Graphics]: Miscellaneous

Keywords: computational fluid dynamics (CFD), software design, systems, applications, flow visualization, vector field visualization

1 Introduction

Demand for visualization solutions for CFD simulation data has grown rapidly in the last decade. This is due, in part, by the interest of manufactures in minimizing the time taken for their production cycle. This objective is realized with the use of CFD software tools to analyze design decisions before constructing real, heavy-weight objects. CFD software can be structured according to the three principle stages typical of engine component design:

1. *modeling*: starting with a model generated by computer aided design (CAD) software, a 3D unstructured mesh is generated consisting of small volumetric cells
2. *simulation*: given the 3D mesh and a set of initial conditions, a simulation of flow through the model is computed
3. *visualization*: the results of the simulation are explored, analyzed, and presented with a variety of visualization tools

*email: {Laramee,Hadwiger,Hauser}@VRVis.at

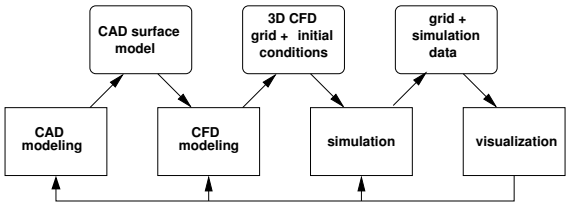

Figure 1: The CFD process is iterative and can be pipelined into modeling, simulation, and visualization stages. Rounded boxes represent input/output data while processing stages are depicted as rectangles.

The process is iterative, as illustrated in Figure 1. The visualization process often either verifies or conflicts the results expected by the engineer and may instigate changes to the model design.

We performed the research and implemented research software inside of a large commercial software package whose job is to perform those tasks conveyed in Figure 1. The result is a system which is further integrated than typical research prototypes. Typical research software consists of stand-alone prototypes for proof-of-concept only. As a result of our integrated system, we enable the possibility to combine multiple visualization options with one another. Incorporating research features into larger systems has both beneficial and non-beneficial consequences. We discuss both the advantages and disadvantages of such an approach.

We focus on the design and implementation of the visualization subsystem shown on the right of Figure 1. More specifically, we focus on those software components that provide geometric and texture-based flow visualization results. We describe several aspects related to the design and implementation of our flow visualization software modules as well as those factors that motivated our decisions.

The rest of this paper is organized as follows: Section 2 describes the three classes of flow visualization techniques that form the basis of design. Section 3 outlines the user requirements and goals of the visualization software. Section 4 presents the overall design of the visualization system while Section 5 details the implementation and design of our flow visualization software modules. Section 6 evaluates some aspects of our design and implementation and discusses some advantages and disadvantages of our work.

2 Flow Visualization Classification

Three different approaches are widely used in flow visualization [Post et al. 2002]:

Direct flow visualization: This category of techniques uses a translation that is as straightforward as possible for representing flow

data in the resulting visualization. The result is an overall picture of the flow. Common approaches are drawing arrows or color coding velocity.

Geometric flow visualization: These approaches often first integrate the flow data and use geometric objects in the resulting visualization. The objects have a geometry that reflects the properties of the flow. Examples include streamlines, streaklines, and timelines. Not all geometric objects are based on integration. Another useful geometric approach is generating isosurfaces, e.g., with respect to an isovalue of pressure or magnitude of velocity. A more thorough description of geometric techniques is presented by Post et al. [Post et al. 2002]

Dense, texture-based flow visualization: A texture is computed that is used to generate a dense representation of the flow. A notion of where the flow travels is conveyed through co-related texture values along the vector field. In most cases this effect is achieved through filtering of texels according to the local flow vector. Texture-based methods offer a dense representation of the flow with complete coverage of the vector field. Examples include Image Based Flow Visualization (IBFV) [van Wijk 2002] and Image Space Advection (ISA) [Laramee et al. 2004b], which can generate both Spot Noise [van Wijk 1991] and LIC-like [Cabral and Leedom 1993] imagery. We note that a full comparison of texture-based flow visualization techniques is beyond the scope of this paper [Laramee et al. 2004a].

The focus of this paper is on the design and implementation of software for the geometric and texture-based category of visualization options provided by our software.

3 System Requirements and Goals

The VRVis research center collaborates with AVL (`www.avl.com`) in order to provide flow visualization solutions for analysis of their CFD simulation result data. AVL's own engineers as well as engineers at industry affiliates use flow visualization software to analyze and evaluate the results of their automotive design and simulation on a daily basis. The analysis of an engineer includes tasks such as searching for areas of extreme pressure, looking for symmetries in the flow, searching for critical points, and comparing simulation results with previous simulation results and with measured, experimental results. As such, AVL engineer's have the following requirements:

Interaction: One pervading message we hear consistently is that users are interested in more interactive control of the flow visualization results–a classic theme in the realm of scientific visualization [Hibbard and Santek 1989]. Users generally want feedback as soon as possible after modifying visualization parameters. Interaction is essential in the engineer's design process. Engineers as well as users from other disciplines are interested in having a collection of user-options and parameters that allow them to fulfill their individual goals, whether their goals are exploration, analysis, or presentation. Interactive tools facilitate an iterative visual analysis and exploration process i.e., an environment in which the user is able to make rapid decisions and refinement based on visualization results.

Platform Independence: Despite the popularity of research relating to programmable graphics hardware, our software design and implementation must maintain platform independence. Our software must function for different users on a wide variety of operating systems including: Linux, HP-UX, SGI, IBM AIX, UNIX, and others. Thus, algorithms or software bound to a specific graphics card are not welcome candidates for inclusion in this system. Platform independence includes not only hardware independence, but software independence as well. That is why all of our software uses only platform independent software libraries such as the well established OpenGL standard.

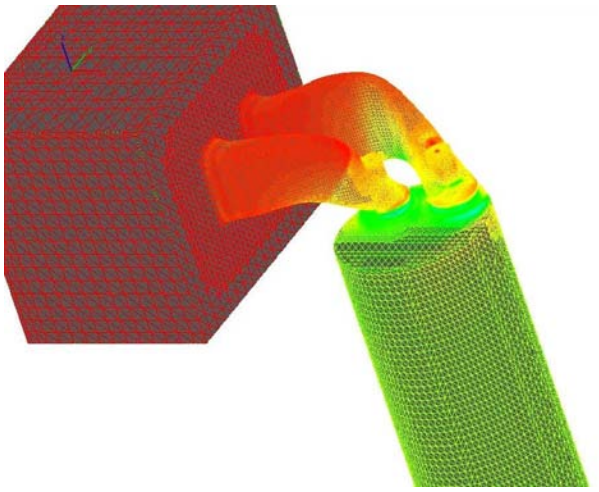

Figure 2: The CFD simulation grid of an intake port. This image illustrates the versatility of a typical, unstructured, CFD simulation grid.

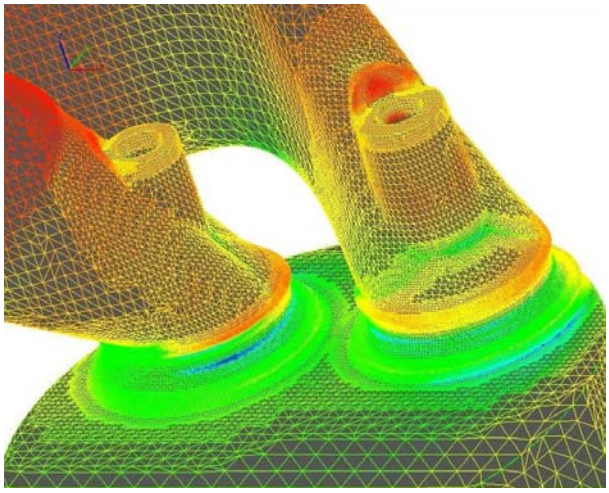

Figure 3: A close-up view of the intake ports in the same CFD simulation grid as shown in Figure 2. The mesh contains multiple, adaptive resolution levels of unstructured grid cells.

Support for a Wide Range of Simulation Data Sets: AVL analyzes a large, varied collection of data sets ranging from small geometries such as small fluid conduits to mid-range size geometries such as cooling jackets, to large geometries such as automotive exteriors. The geometric sizes of these grids differ by *six* or more orders of magnitude as well as the sizes of the underlying polygons. Hence, the tools used to visualize the simulation results also need to span this range of sizes.

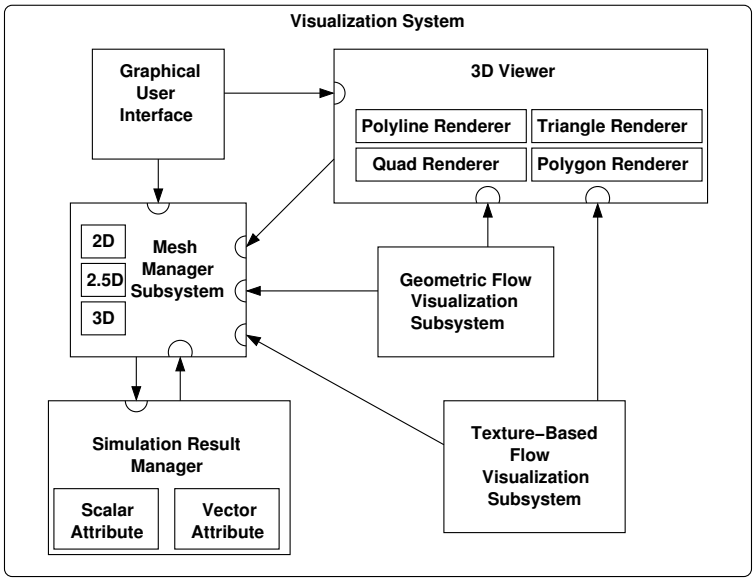

Figure 4: A schematic of the design of the visualization system into which we incorporated our research related software. Only the major subsystems are illustrated.

Support for Versatile CFD Grids: Another reason the users request more interaction control over the visualization results is because CFD meshes embrace a wide variety of components, features, and levels of resolution. To illustrate this idea, we look at Figure 2 showing two intake ports. We observe multiple adaptive levels of resolution: (1) for the flow source on the left and the cylinder on the lower, right, (2) another level of resolution for the connecting pipes in the middle, (3+4) and two levels of resolution for the intake port components. When we look closer (Figure 3) we find five adaptive levels of resolution: (a) two levels for the top of the ports, (b) approximately the same two levels of detail plus an added layer of finer resolution grid cells for a few of the rings around the base of the ports. Facets in the flow source (Figure 2 left) are approximately 1000–2000 times larger than the finest resolution facets at the base of the intake ports.

Tools that Address the Perceptual Challenges in 3D Flow Visualization: Flow visualization on boundary surfaces and in 3D presents additional perceptual challenges such as occlusion, lack of directional cues, lack of depth cues, and visual complexity. Almost all of the CFD simulation models at AVL are unstructured and three dimensional. Although engineers often use 2D cuts through the 3D meshes during their analysis, there is a strong interest in 3D and boundary surface visualization techniques that address the perceptual problems mentioned above. We also know that there is strong evidence to support the notion that users acquire a better understanding of 3D data sets using 3D visualization techniques as opposed to 2D visualization techniques [Ware and Franck 1996].

4 Visualization System Design

The visualization software modules we develop are included in a product called IMPRESS. IMPRESS is part of a larger package called CFDWM (The CFD Workflow Manager) that includes the modeling and simulation modules. In this section, our focus is on the visualization system shown in greater context in Figure 1 and in more detail in Figure 4.

CFDWM is a large project, currently over 4,000 files. Thus, we rely on object-oriented methodology in order to design and incorporate our flow visualization features. In modern, object-oriented software development, more time is spent on software design [Wirfs-Brock et al. 1990] in order to make software more robust, increase code re-use, facilitate maintenance, and make it easier to extend. The design of our software is based on object-oriented methodology.

The design of our visualization system, using the notation of Wirfs-Brock et al. [Wirfs-Brock et al. 1990] is shown in Figure 4. A semicircle with an arrow pointing to it represents a contract. A contract is a subsystem or class interface with other classes or subsystems. It represents the set of services that a subsystem or class provides. Figure 4 illustrates the different subsystems and the relationships they have with one another.

The **Graphical User Interface** subsystem is responsible for presenting all of the user options and associated events triggered by the user. The **3D Viewer** subsystem is responsible for all of the rendering, including point, polyline, triangle, quad, and polygonal primitives. The implementation of the 3D viewer is based on OpenGL for its platform independence. The **Mesh Manager** contains the unstructured CFD mesh. It is responsible for generating slices, surface representations, and volume representations. The **Mesh Manager** has a close relationship with the **Simulation Result Manager** which stores the CFD simulation data attributes such as temperature, pressure, flow velocity etc.

In the next section, we describe two subsystems in more detail: the **Geometric Flow Visualization Subsystem** and the **Texture-Based Flow Visualization Subsystem**. This is where the majority of our research related development was done. These two subsystems, like the others, are composed of a fairly complex set of classes and associated responsibilities.

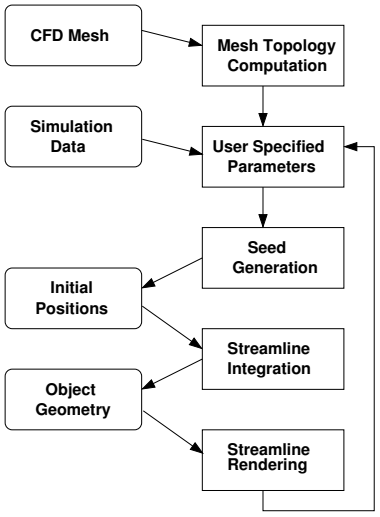

Figure 5: The processing pipeline for the geometric flow visualization subsystem.

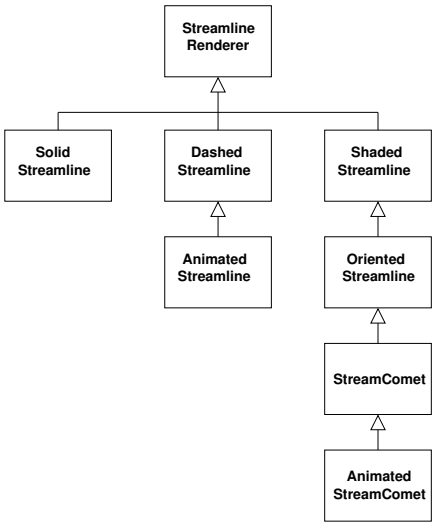

Figure 6: The class hierarchy of streamline and geometric flow visualization options. UML notation is used.

5 Subsystem Design and Implementation

Here we detail how our research software was integrated into a larger visualization system. Our implementation inherits both benefits and non-beneficial aspects of the larger visualization system. It is here we put our design principles into actual practice.

5.1 The Geometric Flow Visualization Subsystem

Figure 5 illustrates the main processing pipeline of the geometric flow visualization subsystem. The input and output data is shown in rectangles with rounded corners and processes are shown in boxes. Note that this design and implementation subsystem focuses on geometric objects such as streamlines which require integration. Other geometric objects such as isosurfaces are handled by another subsystem.

5.1.1 The Geometric Flow Visualization Process

The main input to this process is the CFD mesh and associated vector field data. Since the mesh is unstructured and adaptive resolution, the mesh adjacency information is computed as a preprocessing step. After the user specifies their input requirements, such as the position of the seeding rake or plane and color-mapping parameters, the pipeline follows that of the standard streamline generation process. The 3D seeding process is interactive. This relates back to our requirement of interaction. Engineers require explicit seeding control in order to visualize or highlight specific subsets of the flow. And the subsets of the flow in which the engineers are interested cannot always be found beforehand and detected automatically.

The seeding process involves a grid cell searching phase. The streamlines are then integrated using an Euler integrator with small step sizes for smaller grid cells as the default. The resulting integral paths are stored and handed off to the streamline renderer. Note that an object oriented design like that shown in Figure 5 allows higher order integrators such as a second order Runge-Kutta [Conte and de Boor 1980] to be incorporated into the pipeline with little to no alteration of the other classes. The ability to swap functional components with one another is an important part of big system design.

5.1.2 Geometric Flow Visualization Design

Figure 6 shows our streamline rendering options displayed in the class hierarchy in which they were designed and implemented. The hierarchy, following UML notation [Fowler 2003], illustrates the *is-kind-of* relationship between rendering classes. At the top of the hierarchy we have an abstract base, **Streamline Renderer** that describes the behavior and contains the interface that all streamline rendering objects implement. Animated, shaded, oriented streamlines, and streamcomets all relate back to our requirement of developing tools that address the perceptual challenges in 3D flow visualization since they are all targeted at 3D flow.

Of course one advantage to this type of design is that behavior added to the parent classes are inherited by all of the children. Thus adding features such a streamrunner [Laramee 2002], color-mapping, and anti-aliasing can be inherited by child classes who, in many cases, may automatically pick up the new features. Also, adding new rendering features only requires a few lines of new code to be written since we only have to override the render method of a parent class. We note also, that the streamline computation, e.g., Euler and Runge-Kutta integrators on 2D slices, 2.5D surfaces, and 3D meshes, are completely separate from the rendering subsystem. Hence any rendering option can be associated with any integration result.

5.2 The Texture-Based Flow Visualization Subsystem

The texture-based flow visualization subsystem is where the most research related software development took place. Three new, closely related algorithms were implemented, namely Image-Based Flow Visualization (IBFV) [van Wijk 2002], Image Space Advection (ISA) [Laramee et al. 2003], and Image Based Flow Visualization for Curved Surfaces (IBFVS) [Laramee et al. 2004b]. The ISA

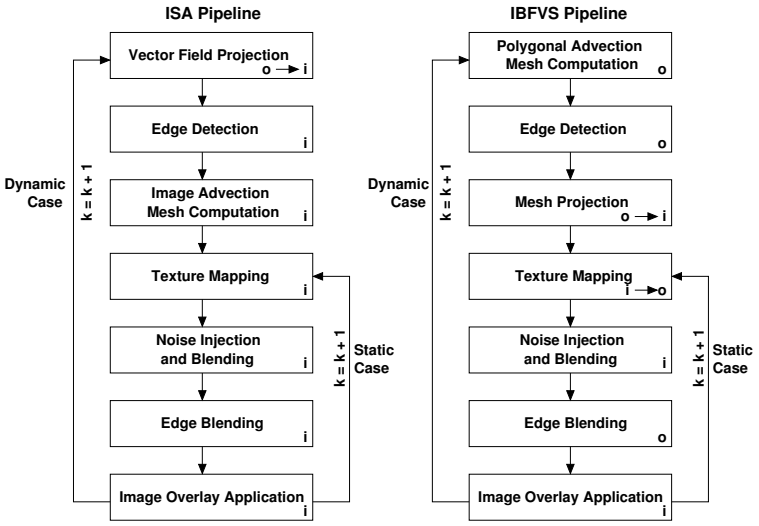

Figure 7: The processing pipeline of the texture-based flow visualization subsystem.

and IBFVS algorithms were implemented within the same software package in order to compare them with one another.

5.2.1 The Texture-Based Flow Visualization Process

A side-by-side illustration of the processing pipelines of both algorithms is shown in Figure 7. In brief, the ISA and IBFVS algorithms simplify the problem of advecting textures on surfaces by confining the advection of texture properties to image space. After a projection to image space phase, a series of textures are mapped, blended, and advected. The ISA method for visualization of flow on surfaces is comprised of the following procedure (Figure 7, left): (1) project the vector field to the image plane, (2) detect geometric edge discontinuities, (3) compute advected texture coordinates, (4) advect the image, (5) inject and blend in noise, (6) blend additional noise along geometric edge discontinuities, and (7) apply shading and other additional graphics. The IBFVS method is very similar, the essential difference being that advected texture coordinates are computed in object space rather than image space. Steps 1-7 of the pipeline are necessary for the dynamic cases of time-dependent geometry, rotation, translation, and scaling, and only a subset is needed for the static cases (steps 4-7) involving no changes to the view-point and steady-state flow. Each stage is described in more detail in previous research [Laramee et al. 2004b].

In order to speed up the computation time of advecting textures on surfaces, texture coordinates are computed in image space rather than 3D. The result is that some portions of the algorithms take place in image space and some in object space. Those operations which take place in image space are notated with an **i** in Figure 7, similarly an **o** for those operations taking place in object space. In some pipeline modules, like the ISA vector field projection, a transition takes between object space and image space. This is notated with $\mathbf{o} \longrightarrow \mathbf{i}$. Which stages of the respective pipelines take place in image space and object space identify the essential differences between the algorithms, since conceptually they share many overlapping components.

Another property that makes these algorithms faster than previous related work is that the stages of the pipeline shown in Figure 7 map well to graphics card hardware. However, rather than depending on a specific type graphics card, these algorithms exploit only standard features offered by graphics cards that support OpenGL 1.1, thus making them fast across a variety of platforms.

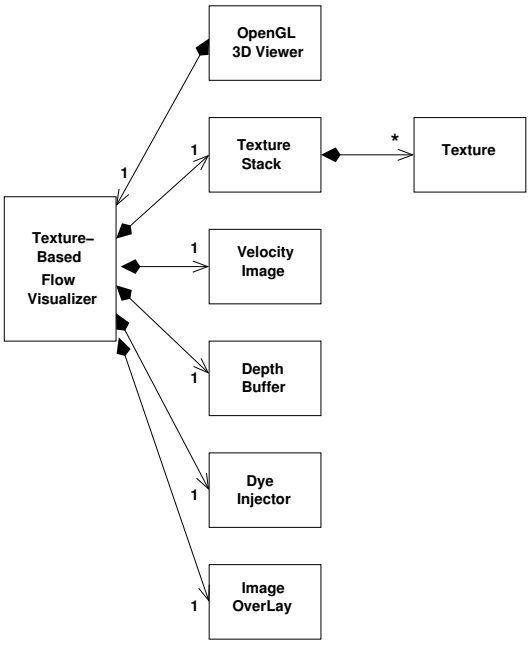

Figure 8: The major components of the texture-based flow visualization design. Here aggregation, or *is-part-of*, relationships are shown.

5.2.2 Texture-Based Flow Visualization Design

Figure 8 shows the class relationship between the major components of the texture-based flow visualization subsystem, again using UML notation. However, rather than showing is-kind-of relationships as in Figure 6 we show *composition*, a variety of aggregation. With composition, the part object may belong to only one whole, further, the parts are usually expected to live and die with the whole [Fowler 2003]. For example, the **Texture Stack** object is part

of the **Texture-Based Flow Visualizer** object and the relationship is one-to-one. Furthermore, an instance of **Texture Stack** may be in an instance of **Texture-Based Flow Visualizer** but not the other way around. This is indicated by the black diamond shape arrow. It is in fact possible to create multiple instances of the texture-based flow visualizer, just like it is possible to create more than one instance of a viewer. However, this may result in a performance hit as multiple instances of these classes compete for hardware resources.

Here we outline the major components that make up the texture-based flow visualization subsystem shown in Figure 8. The **Texture-Based Flow Visualizer** is the class with the most responsibility, namely that of coordinating pipelines in Figure 7 of both the ISA and IBFVS algorithms [Laramee et al. 2004b]. The **OpenGL 3D Viewer** class is responsible for general rendering of primitives such as points, lines, and polygons. The **Texture Stack** is responsible for managing a stack of textures. This stack can be used to implement the injection and blending of noise for the IBFV [van Wijk 2002], ISA [Laramee et al. 2003], and IBFVS [Laramee et al. 2004b] algorithms. The **Texture Stack** is composed of individual **Textures**. It is worthy of note that textures are also an object in OpenGL 1.1. A **Velocity Image** is responsible for the vector field projection, the first step in the ISA pipeline of Figure 7 which simplifies the computation from 3D to 2D. The **Depth Buffer** object stores a copy of the OpenGL depth buffer. This information is used in the edge detection and blending process in the ISA algorithm [Laramee et al. 2003]. IBFV, ISA, and IBFVS can all include **Dye Injector** functionality. The representation of the dye injection design has been simplified here. In fact it is its own subsystem including a hierarchy of **Dye Source** objects. The **Image OverLay** includes perceptual information such as shading and depth cues and is the final stage of both ISA and IBFVS. Here, we use a separate object for this job.

Laying out the responsibilities in this way facilitates improvement. Immediately we can see one area of improvement would be to split up the responsibilities of the **Texture-Based Flow Visualizer** into two separate classes, one for only the ISA pipeline and another for IBFVS, perhaps with a common base class. Figures 6 and 8 are simplified representations of the overall design. They leave out the classes responsible for the user interface and it's associated event handlers. A design process is essential for producing stable software that is robust enough to meet the requirements of a commercial grade application.

Our commercial system targets many different hardware architectures. As we know, hardware evolves rapidly, especially graphics hardware. The range of hardware our application targets also spans several years, including systems that are more than five years old. This is one reason we have given users interactive control of the amount of texture-memory used by the texture-based flow visualizer. Users with old systems may set the controls to use less texture memory for faster performance also trading off quality, or more texture memory for users with modern graphics cards and plenty of texture memory.

6 Discussion and Evaluation

After presenting the design and implementation of our geometric and texture-based flow visualization sub-systems, we now discuss the advantages and disadvantages of implementing them into an integrated system and evaluate the modules against the requirements and goals specified in Section 3.

A big advantage of integrating the research related subsystems into a larger commercial system is the ability to combine visualization options. Figure 9 shows the visualization of tumble flow [Laramee et al. 2004c] using a combination of texture-based flow visualization, color-mapping, streamlines seeded with two seeding planes, and a color-mapped pressure isosurface [1]. Figure 10 shows our application including the user interface components. It is unusual to have this many visualization options combined into a commercial software package, and even more rare in a research prototype. Providing engineers and other users with a wide variety of options is helpful because each technique has a unique set of advantages and disadvantages, e.g., some techniques are better suited for 3D visualization than others. Another advantage of a big project is that much functionality has already implemented, especially the routine engineering tasks such as file I/O, saving and loading data sets, setting up a general purpose GUI (Figure 10), acquiring CFD grids and simulation data etc. These are often tasks which a researcher must dedicate time towards in order to build a good prototype.

Naturally there are also disadvantages to integrating research related software into a large industry level system. Since large systems can be composed of thousands of files and classes, the time taken to learn and understand the software well enough to add new modules is longer. Common, daily developer tasks also require more time for the developer of a large system. Compiling the entire CFD workflow manager requires more than one hour. Simply loading the project source into main memory over a network can take five to ten minutes.

Plus there is also overhead from testing. The software is used by more people, hence it should be more stable. Features must be robust enough to analyze a very wide variety of data sets, not just two or three data sets carefully selected by the researcher. However, coupled directly with this is an advantage: the large number of data sets that require exploration and analysis force the software engineer to write algorithms which are robust and efficient.

With respect to interaction, many of our visualization features are interactive. Our animated streamlines achieve real-time frame rates. In texture-based flow visualization, ISA and IBFVS are amongst the first texture-based flow visualization algorithms to achieve interactive frame rates for surfaces. Of course, when the data set sizes grow big enough, interactivity becomes problematic. Also, our streamline integration process is not interactive for large numbers of streamlines, only the rendering phase. Platform independence is achieved through the use of platform independent libraries. To our knowledge, OpenGL is the only widely supported, platform independent graphics library. Most of our features rely on OpenGL 1.1. Also, we use the FOX windowing toolkit (www.fox-toolkit.org) in order to achieve platform independence with respect to the user interface. FOX is an easy-to-learn GUI library suitable for CFD applications. FOX was the programming library of choice (as opposed to other libraries like the VTK [Kit], Amira [TGS], or AVS Express [Adv]) for multiple reasons including: (1) it is platform independent, (2) it is open source, (3) it is developed specifically for commercial CFD applications. As such it was an appropriate choice.

In terms of versatility, our tools, having been incorporated into commercial software, must undergo more testing by more users than typical research prototypes. Our features have been used to explore and visualize on a wide range of data sets with both static and dynamic geometry. One important aspect of the design is a complete range of user control. It is important to provide the user with control over the visualization parameters in order to meet the versatility and range of data sets. The developer simply cannot predict all of the models and their respective features to which the visualization

[1] For supplementary, high resolution images, please visit: `http://www.VRVis.at/scivis/design/`

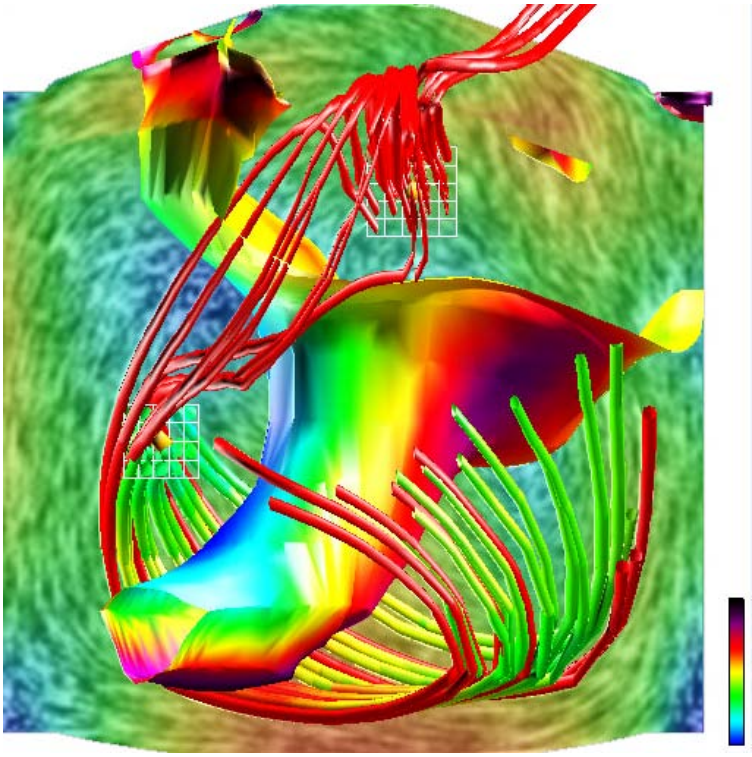

Figure 9: Visualization of tumble motion using a combination of several visualization options including: color-mapping, slicing, texture-based flow visualization, isosurfacing, and streamlines seeded with multiple seeding planes.

techniques will be applied.

Our range of tools also includes those that address the perceptual challenges in 3D visualization. These tools include the streamrunner [Laramee 2002], streamcomets, and variable resolution streamline seeding plane [Laramee et al. 2004c]. However, applying texture-based flow visualization techniques to true 3D flow still remains an unsolved problem in this context.

7 Conclusion

We have presented the design and implementation of research based software modules integrated within a larger, industry level visualization system. We have discussed our design decisions and the associated motivation for those decisions. And although we have focused on flow visualization specific software, we believe the principles outlined here can be applied in a more general way to other similar projects. The result of incorporating research related software into a large system brings both advantages and disadvantages. Benefits include a rich visualization feature set and robustness while disadvantages include all those tasks inherent in commercial software development such as a steep learning curve and large project maintenance.

8 Acknowledgments

We would like to thank all those who have contributed to financing this research, including AVL (`www.avl.com`) and the Austrian research program Kplus (`www.kplus.at`). All CFD simulation data is courtesy of AVL.

References

ADVANCED VISUAL SYSTEMS INC. *OpenVIZ.* 300 Fifth Avenue, Waltham, MA 02451. http://www.avs.com.

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging Vector Fields Using Line Integral Convolution. In *Poceedings of ACM SIGGRAPH 1993*, Annual Conference Series, 263–272.

CONTE, S. D., AND DE BOOR, C. 1980. *Elementary Numerical Analysis*. McGraw-Hill, New York.

FOWLER, M. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, third ed. Object Technology Series. Addison-Wesley, Sept.

HIBBARD, W., AND SANTEK, D. 1989. Interactivity is the Key. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, 39–43.

KITWARE INC. *The Visualization Toolkit*. 28 Corporate Drive, Suite 204, Clifton Park, New York 12065. http://www.kitware.com.

LARAMEE, R. S., JOBARD, B., AND HAUSER, H. 2003. Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proceedings IEEE Visualization '03*, IEEE Computer Society, 131–138.

LARAMEE, R. S., HAUSER, H., DOLEISCH, H., POST, F. H., VROLIJK, B., AND WEISKOPF, D. 2004. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum 23*, 2 (June), 203–221.

LARAMEE, R. S., VAN WIJK, J. J., JOBARD, B., AND HAUSER, H. 2004. ISA and IBFVS: Image Space Based Visualization

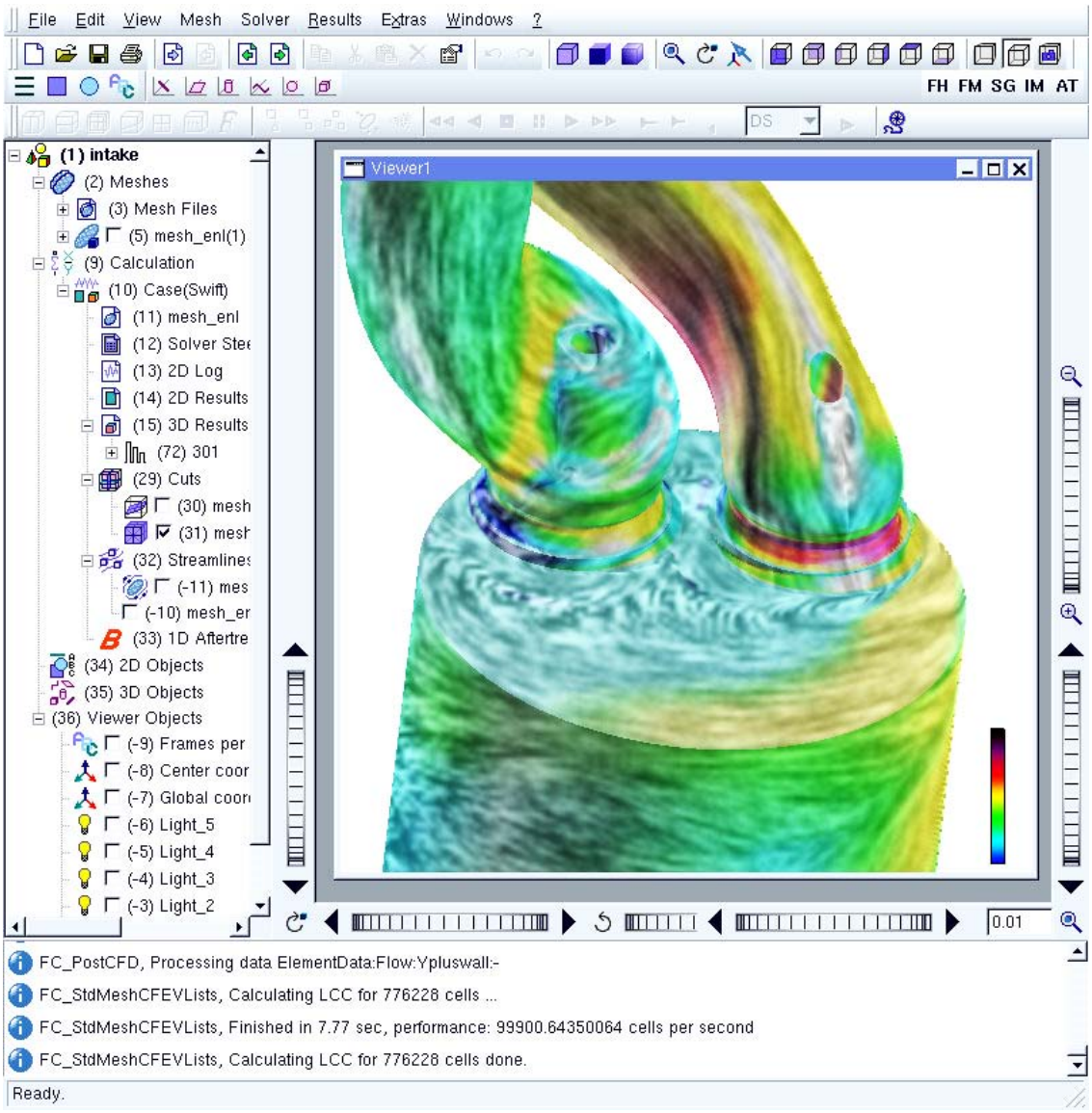

Figure 10: A screen shot of our industry level application being used to visualize the vector field at the surface of two intake ports.

of Flow on Surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 6 (Nov.), 637–648.

LARAMEE, R. S., WEISKOPF, D., SCHNEIDER, J., AND HAUSER, H. 2004. Investigating Swirl and Tumble Flow with a Comparison of Visuaization Techniques. In *Proceedings IEEE Visualization '04*, 51–58.

LARAMEE, R. S. 2002. Interactive 3D Flow Visualization Using a Streamrunner. In *CHI 2002, Conference on Human Factors in Computing Systems, Extended Abstracts*, ACM Press, ACM SIGCHI, 804–805.

POST, F. H., VROLIJK, B., HAUSER, H., LARAMEE, R. S., AND DOLEISCH, H. 2002. Feature Extraction and Visualization of Flow Fields. In *Eurographics 2002 State-of-the-Art Reports*, 69–100.

TGS INC. (TEMPLATE GRAPHICS SOFTWARE). *Amira 3.1, User's Guide and Reference Manual*. 5330 Carroll Canyon Road, Suite 201, San Diego, California 92121-3758. http://www.amiravis.com.

VAN WIJK, J. J. 1991. Spot noise-Texture Synthesis for Data Visualization. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, T. W. Sederberg, Ed., vol. 25, 309–318.

VAN WIJK, J. J. 2002. Image Based Flow Visualization. *ACM Transactions on Graphics 21*, 3, 745–754.

WARE, C., AND FRANCK, G. 1996. Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions. *ACM Transactions on Graphics 15*, 2 (Apr.), 121–140.

WIRFS-BROCK, R., WILKERSON, B., AND WIENER, L. 1990. *Designing Object-Oriented Software*. Prentice-Hall.