DISSERTATION

INTERACTIVE 3D FLOW VISUALIZATION BASED ON TEXTURES AND GEOMETRIC PRIMITIVES

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von

Priv.-Doz. Dipl.-Ing. Dr.techn. Helwig Hauser,
VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH,

eingereicht an der Technischen Universität Wien,
Fakultät für Informatik

von

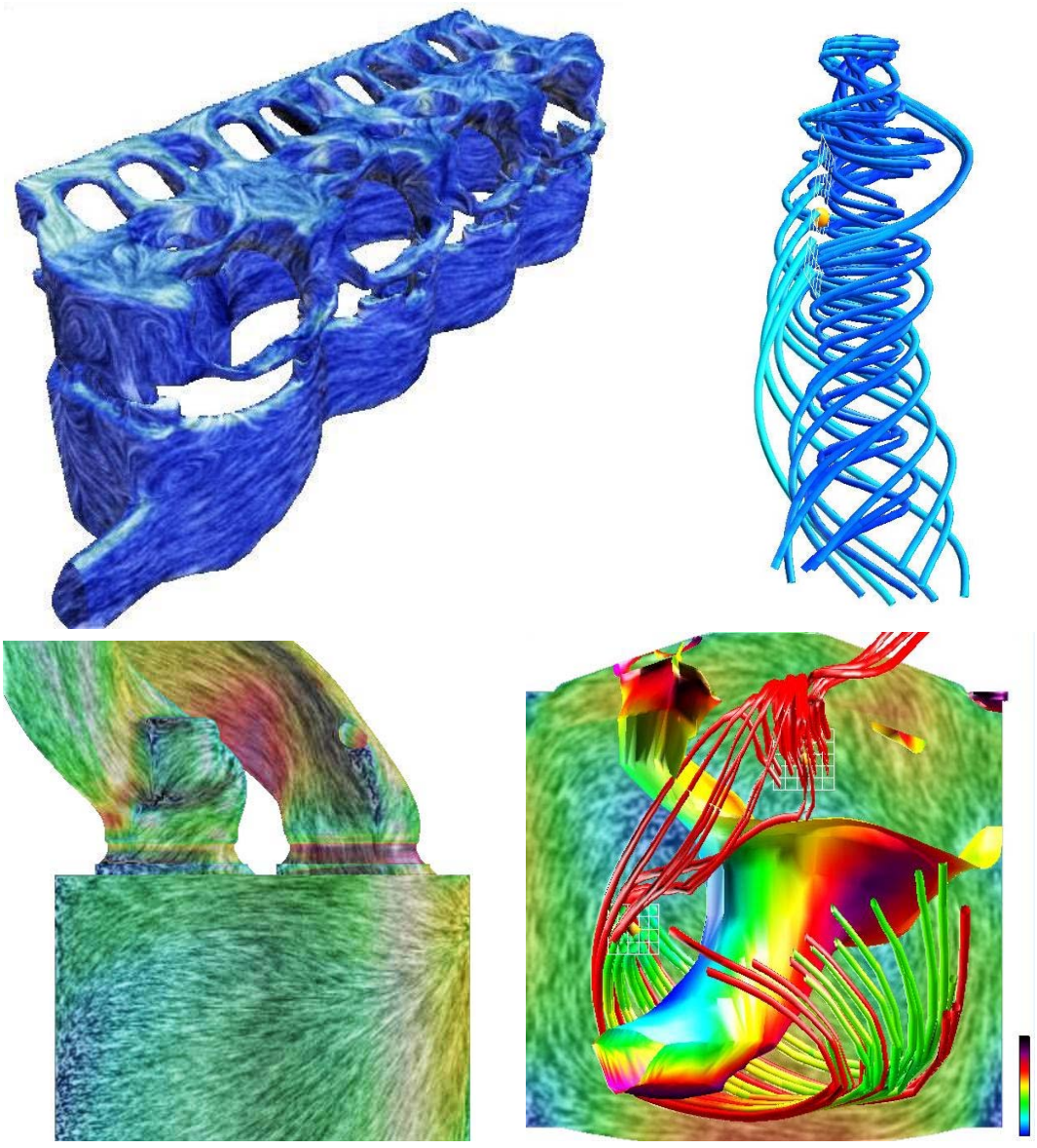
Robert S. Laramee, MSc.
Matrikelnummer 0327742
Mariengasse 5, 8020 Graz
Österreich

Wien, im Dezember 2004

Robert S Laramee

Interactive 3D Flow Visualization Based on Textures and Geometric Primitives

PhD Thesis



mailto:Laramee@VRVis.at
http://www.VRVis.at/ar3/pr2/laramee/

Abstract

This thesis presents research in the area of flow visualization. The theoretical framework is based on the notion that flow visualization methodology can be classified into four main areas: direct, geometric, texture-based, and feature-based flow visualization. Our work focuses on the direct, geometric, and texture-based categories, with special emphasis on texture-based approaches.

After presenting the state-of-the-art, we discuss a technique for resampling of CFD simulation data. The resampling tool addresses both the perceptual problems resulting from a brute force hedgehog visualization and flow field coverage problems. These challenges are handled by giving the user control of the resolution of the resampling grid in object space and giving the user precise control of where to place the vector glyphs.

Afterward, we present a novel technique for visualization of unsteady flow on surfaces from computational fluid dynamics. The method generates dense representations of time-dependent vector fields with high spatio-temporal correlation. While the 3D vector fields are associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space. Frame rates of up to 60 frames per second are realized by exploiting graphics card hardware. We apply this algorithm to unsteady flow on boundary surfaces of, large, complex meshes from computational fluid dynamics composed of more than 200,000 polygons, dynamic meshes with time-dependent geometry and topology, as well as medical data. We also apply texture-based flow visualization techniques to isosurfaces. The result is a combination of two well known scientific visualization techniques, namely iso-surfacing and texture-based flow visualization, into a useful hybrid approach.

Next we describe our collection of geometric flow visualization techniques including oriented streamlines, streamlets, a streamrunner tool, streamcomets, and a real-time animated streamline technique. We place special emphasis on necessary measures required in order for geometric techniques to be applicable to real-world data sets.

In order to demonstrate the use of all techniques, we apply our direct, geometric, and texture-based flow visualization techniques in order to investigate swirl and tumble motion, two flow patterns found commonly in computational fluid dynamics (CFD). Our work presents a visual analysis of these motions across three spatial domains: 2D slices, 2.5D surfaces, and 3D.

Kurzfassung

Diese Dissertation stellt aktuelle Forschungsergebnisse aus dem Bereich der Strömungsvisualisierung vor. Der theoretische Rahmen baut auf der Erkenntnis auf, dass man grob vier verschiedene Arten von Strömungsvisualisierungsansätzen unterscheiden kann: direkte, geometrische, Textur-basierte und Merkmal-basierte. Diese Arbeit setzt sich vor allem mit direkter, geometrischer und Textur-basierter Strömungsvisualisierung auseinander, wobei das Hauptaugenmerk auf Letzterer liegt.

Nach einem Überblick über den momentanen Stand der Forschung wird eine Methode zum Resampling von Strömungssimulationsdaten (CFD-Daten) vorgestellt. Dieser Ansatz versucht dabei sowohl Wahrnehmungsprobleme, die bei einer direkten Anwendung der Hedgehog Visualisierugstechnik auftreten, als auch Unzulänglichkeiten bei der Abdeckung des Strömungsfeldes zu beheben. Das wird erreicht, indem dem Benutzer bzw. der Benutzerin die genaue Kontrolle über die Auflösung des Resampling Gitter im Objektraum und über die Platzierung der Vektorglyphen überlassen wird.

Wir stellen eine neuartige Methode zur Visualisierung von nicht-stationären Strömungssimulationsdaten auf Oberflächen vor. Während die dreidimensionalen Vektorfelder beliebige, durch Dreiecksnetze gegebene Oberflächen haben können, beschränkt sich die Generierung und Advektion von Textureigenschaften auf den Bildraum. Der Einsatz von Graphik-hardware ermöglicht Frameraten von bis zu 60 Frames pro Sekunde. Die Geometrie, auf deren Aussenflächen die nicht-stationären Strömungsdaten gegeben sind, auf die wir diesen Algorithmus anwenden, umfasst große, komplexe Dreiecksnetze mit mehr als 200.000 Polygonen, wobei sich Geometrie und Topologie über die Zeit verändern können, sowie auch Datensätze aus der Medizin. Wir wenden Textur-basierte Strömungsvisualisierungstechniken auch für Isoflächen an. Das Ergebnis ist die Verschmelzung zweier bekannter Methoden aus dem Bereich der wissenschaftlichen Visualisierung–nämlich iso-surfacing und Textur-basierte Strömungsvisualisierung–in einen nützlichen Hybridansatz.

Danach wendet sich die Arbeit einer Reihe von Techniken zur geometrischen Strömungsvisualisierung zu, unter anderem gerichtete Strömungslinien, Streamlets, einem Streamrunner Werkzeug, Streamcomets und einer Methode für in Echtzeit animierte Strömungslinien. Besondere Beachtung wird dabei Maßnahmen geschenkt, die notwendig sind, um geometrische Techniken auch für praxisnahe Datensätze anwendbar zu machen.

Abschließend wenden wir direkte, geometrische und Textur-basierte Strömungsvisualisierungsmethoden an, um Strudel- und Wirbelbewegungen zu untersuchen–zwei Strömungsmuster, die oft in Strömungssimulationsdaten anzutreffen sind. Die Arbeit stellt eine visuelle Analyse dieser Bewegungen in den drei räumlichen Ausprägungen vor: 2D Schichten, 2.5D Oberflächen und 3D [1].

[1] Thanks to Harald Piringer and Helmut Doleisch for help with the translation

Related Publications

This thesis is based on the following publications:

- Robert S. Laramee, Bruno Jobard, and Helwig Hauser, **Image Space Based Visualization of Unsteady Flow On Surfaces** in *Proceedings of* IEEE *Visualization (Vis 2003)*, pages 131–138, October 19–24, 2003, Seattle, Washington
- Robert S. Laramee, **FIRST: A Flexible and Interactive Resampling Tool for CFD Simulation Data** in *Computers & Graphics, Vol. 27, No. 6*, pages 905–916, 2003
- Robert S. Laramee, Jürgen Schneider, and Helwig Hauser, **Texture-Based Flow Visualization on Isosurfaces from Computational Fluid Dynamics** in *Proceedings of the* 6^{th} *Joint* IEEE *TCVG–* EUROGRAPHICS *Symposium on Visualization (VisSym 2004)*, pages 85–90, May 19–21, 2004, Konstanz, Germany
- Robert S. Laramee, Daniel Weiskopf, Jürgen Schneider, and Helwig Hauser, **Investigating Swirl and Tumble Flow with a Comparison of Visualization Techniques** in *Proceedings of* IEEE *Visualization (Vis 2004)*, pages 51–58 October 15–19, 2004, Austin, Texas,
- Robert S. Laramee, Jarke J. van Wijk, Bruno Jobard, and Helwig Hauser, **ISA and IBFVS: Image Space Based Visualization of Flow on Surfaces** in IEEE *Transactions on Computer Graphics and Visualization (*IEEE *TVCG), Vol. 10, No. 6*, November/December 2004, pages 637–648
- Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post, and Daniel Weiskopf, **The State of the Art in Flow Visualization: Dense and Texture-Based Techniques** in *Computer Graphics Forum, Vol. 23, No. 2*, 2004, pages 203–221
- Robert S. Laramee, **Interactive 3D Flow Visualization Using a Streamrunner** in *CHI 2002 Conference on Human Factors in Computing Systems, Extended Abstracts*, pages 804–805, 20–25 April 2002, Minneapolis, Minnesota
- Frits H. Post, Benjamin Vrolijk, Helwig Hauser, Robert S. Laramee, and Helmut Doleisch, **Feature Extraction and Visualization of Flow Fields** in EUROGRAPHICS *2002, State of the Art Reports*, pages 69–100, September 4–6 2002, Saarbrücken, Germany
- Frits H. Post, Benjamin Vrolijk, Helwig Hauser, Robert S. Laramee, and Helmut Doleisch, **The State of the Art in Flow Visualisation: Feature Extraction and Tracking** in *Computer Graphics Forum, Vol. 22, No. 4*, 2003, pages 775–792

Contents

Abstract i

Kurzfassung ii

Related Publications iii

1 Introduction: Scientific Visualization **2**

1.1 Flow Visualization 4

1.2 Classification and Contribution 5

1.3 Overview 7

2 Flow Visualization, the State of the Art **8**

2.1 Spatial, Temporal, and Data Dimensionality 8

2.2 Data Sources 9

2.3 Fundamentals 10

2.4 Direct Flow Visualization 12

2.5 Dense and Texture-Based Flow Visualization 14

2.6 Geometric Flow Visualization 27

2.7 Comparisons and Discussion 32

2.8 Discussion and Future Prospects 32

3 FIRST: Flexible and Interactive ReSampling **34**

3.1 Flexible Tools for Unstructured Grids 34

3.2 Related Work in Resampling 36

3.3 Interactive Visualization and Analysis 38

3.4 FIRST: A Flexible and Interactive ReSampling Tool 38

3.5 Results 43

3.6 Discussion and Future Work 44

4 ISA: Image Space Based Visualization of Unsteady Flow on Surfaces 47

4.1 Physical Space vs. Parameter Space vs. Image Space 48
4.2 Method Overview 50
4.3 Vector Field Projection 50
4.4 Advection Mesh Computation and Boundary Treatment 53
4.5 Edge Detection and Blending 53
4.6 Noise Blending 54
4.7 Image Overlay Application 56
4.8 Implementation 56
4.9 Performance and Results 57
4.10 Discussion and Future Work 58

5 Texture-Based Flow Visualization on Isosurfaces 62

5.1 Isosurfaces 63
5.2 Applying Texture-Based Flow Visualization 63
5.3 Method Background 64
5.4 Texture-Based Flow Visualization on Isosurfaces 64
5.5 Results and Discussion 69
5.6 Performance 70
5.7 Discussion and Future Work 70

6 Geometric Flow Visualization Techniques 72

6.1 The Versatility of CFD Grids 72
6.2 Perceptual Challenges 73
6.3 Oriented Streamlines and Streamlets 74
6.4 Animated Streamlines 75
6.5 Streamcomets 77
6.6 Results 79
6.7 Discussion and Future Work 81

7 Investigating Swirl and Tumble Motion 82

7.1 Evaluating Swirl and Tumble Motion 82
7.2 Visualizing Flow Motion on 2D Slices 84
7.3 Swirl and Tumble Flow Visualized on Surfaces 86
7.4 3D and Hybrid Approaches 88
7.5 Trade-Offs 92

7.6 Discussion and Future Work 93

8 Software Design and Implementation 95

8.1 System Requirements and Goals 96

8.2 Visualization System Design 97

8.3 User Interface Design 101

8.4 Discussion and Evaluation 106

9 Summary 109

9.1 Resampling of CFD Simulation Data 110

9.2 ISA: Image Space Based Visualization of Unsteady Flow on Surfaces 111

9.3 Texture Based Visualization of Flow on Isosurfaces 113

9.4 Geometric Flow Visualization Techniques 114

9.5 Investigating Swirl and Tumble Motion 116

9.6 Discussion 118

Conclusions 119

Bibliograpy/References 120

Curriculum Vitae 131

Appendix: Bridging the Gap Between Industry and Research 135

Index 137

Chapter 1

Introduction: Scientific Visualization

*"The code **is** the documentation."*

– Dan R. Lipsa [1] (1970–)

Here we search for a logical starting point for a thesis on flow visualization. We start by using the knowledge hierarchy map shown in Figure 1.1. At the top of the hierarchy, we find *Computer Science*, currently a core topic of study at most universities. Computer science itself covers several disciplines, only a sampling of which is shown in the knowledge hierarchy. As we move down the hierarchy, the topics become more focused. The path of this thesis is illustrated by the relations down the middle. This thesis is concerned with the topic of *Flow Visualization*, a classic sub-topic of scientific visualization.

According to the Merriam-Webster dictionary (online at `www.m-w.com`), visualization is defined as *(1)* the formation of mental visual images, *(2)* the act or process of interpreting in visual terms or of putting into visible form, *(3)* the process of making an internal organ visible by the introduction (as by swallowing, by an injection, or by an enema) of a radiopaque substance followed by roentgenography. Already, the dictionary points to an important application of visualization, namely, medical visualization.

Scientific visualization, sometimes called data visualization,

[1] Romanian software engineer and educator, Dan (Radu) was a colleague of mine in the PhD program at the University of New Hampshire. After UNH he joined Ecora (`www.ecora.com`) from which this quote was inspired. Afterward, he joined the staff at Armstrong Atlantic State University in Georgia.

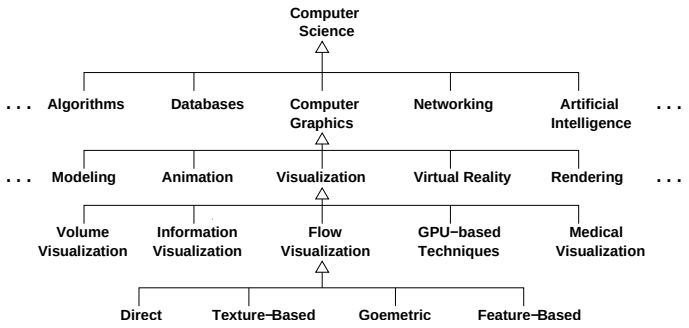


Figure 1.1: A knowledge hierarchy that places the subject of this thesis into a larger context of subjects.

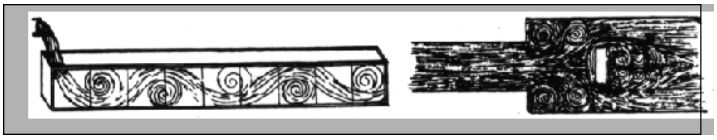


Figure 1.2: An early, hand-drawn visualization by Leonardo da Vinci. Images from *Frontiers of Scientific Visualization* by Pickover and Tewksbury [113].

is the transformation of data, usually numerical data, into images. This process was done by hand a long time ago [113], as illustrated in Figure 1.2, drawn by Leonardo da Vinci (1452-1519). We are concerned with computer-aided visualization with computer generated data.

Data can come from a wide variety of sources such as simulation, modeling, measurments, or from non-scientific disciplines such as finance, marketing, and business. The goal of visualization is to gain a deeper understanding of data. Visualization allows us to see structures and find patterns that are unable to see from a vast array of raw numbers. Some (but not all) of the classic sub-topics of visualization are :

- **volume visualization:** an methodology for visualizing 3D data that may use discrete polygonal primitives or volume rendering techniques. Volume rendering is based on the voxel primitive. Data sources often come from the medical domain, e.g., computer aided tomography (CAT) or magnetic resonance imaging (MRI).
- **information visualization:** assigns an abstract geometry or topology to data that does not already have an inherent geometric representation. Data sources are often financial or economic in nature. Example visualization techniques include the use of pie charts, scatter plots, and parallel coordinates [172].
- **GPU-based techniques:** a rapidly growing area of research is centered around programmable GPUs. The goal is to speed up computation, that might otherwise take place on the CPU by taking advantage of the computing features offered by the GPU.
- **medical visualization:** this is the area that many people intuitively connect with the field of visualization. Medical visualization techniques illustrate subsets of the human body, such as the skeleton or brain, using data generated by medical tools such as CAT scanners. Clearly advances in medicine are a strong driving factor for innovation in this field.

Figure 1.3 shows a recent visualization result that overlaps three of these categories, namely, volume rendering, GPU programming, and medical visualization [49]. We note that Figure 1.1 is not meant to be taken too literally. In fact, it would be very difficult to find two researchers who would agree to a clean classification. We chose this classification based on simplicity and experience. There are conferences and/or workshops dedicated to each one of these topics. The IEEE Visualization conference also divides up literature into the visualization categories shown.

Of course trying to place visualization into a map such as Figure 1.1 is inherently risky. This is because visualization overlaps with many other disciplines such as computer graphics, imaging, statistics, computational geometry, numerical analysis, and studies in human perception. Figure 1.1 is merely an attempt to place our core topic into the big picture. We also note that the levels in the hierarchy can continue down to subjective levels of specialization. For example, all of the flow visualization topics can be further divided up by dimensionality both spatial and temporal, or other classifications, e.g., see Chapter 2.

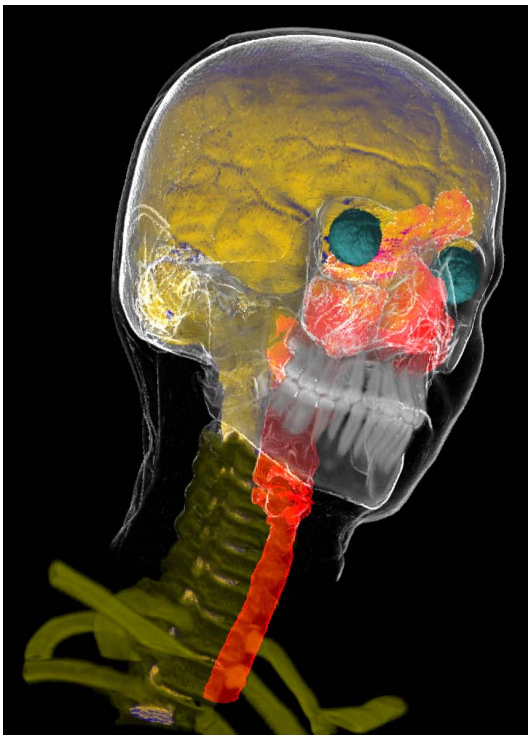


Figure 1.3: A visualization result that involves elements from volume rendering, GPU-based programming, and medical visualization [49].

1.1 Flow Visualization

Flow visualization is the visualization of data with a magnitude *and direction*, with special emphasis here on direction. Vector data often results from the study of fluid flow or a derivative quantity.

The most obvious way of visualizing vector data is with the use of glyphs. However, the use of glyphs is associated with several problems including (1) occlusion, (2) visual complexity, (3) problems with placement, either too sparse or too dense, (4) problems with interpretation [158], (5) lack of spatial coherency, and more. A more detailed discussion of these problems is given in Chapter 3. As a result, a lot of work has been done in the area of vector field visualization.

A good starting point for the topic flow visualization is to look at the data. Velocity, $\mathbf{v} = d\mathbf{x}/dt$, is itself a derivative quantity. If we imagine tracking a massless particle through a vector field, then the displacement of such a point can be described simply by:

$$d\mathbf{x} = \mathbf{v}\, dt \tag{1.1}$$

In order to evaluate equation 1.1, we can express it in integral form:

$$\mathbf{x}(t, \mathbf{x}_0) = \mathbf{x}_0 + \int_{\tau=0}^{t} \mathbf{v}(\tau)\, d\tau \tag{1.2}$$

This is one of the most fundamental equations in the field of flow visualization. Generally, all geometric flow visualization techniques [116], texture-based techniques [82], and feature-based approaches [117] relate back to equation 1.2 in some way.

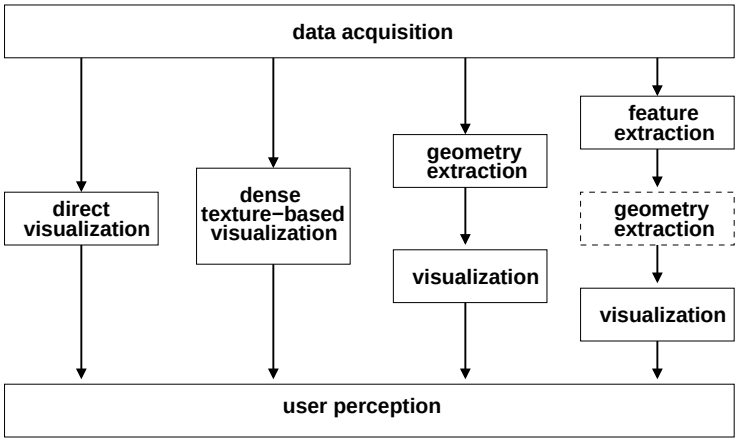


Figure 1.4: Classification of flow visualization techniques – (left) direct, (middle-left) texture-based, (middle-right) based on geometric objects, and (right) feature-based.

For the data we are concerned with, namely CFD simulation data, equation 1.2 cannot be solved analytically. And like most flow visualization researchers, we rely on numerical integration techniques as an approximation. The simplest and perhaps most common approximation of equation 1.2 is given by Euler's method:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i \, \Delta t \tag{1.3}$$

where the position of a massless particle at time $i+1$ is given by the sum of the previous position and the product of the velocity times an incremental time step Δt. One problem with using Euler integration is its inherent error of order $O(\Delta t^2)$, which may not be accurate enough in some cases [135]. Hence there are more accurate (higher order) alternatives such as the second order Runge-Kutta integrator [22]:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{\Delta t}{2}(\mathbf{v}_i + \mathbf{v}_{i+1}) \tag{1.4}$$

where velocity $\mathbf{v}_{i+1}$ is computed using equation 1.3. The error of equation 1.4 is $O(\Delta t^3)$ [22]. When compared with equation 1.3, equation 1.4 allows us to use a larger integration step for the cost of one additional function evaluation. Higher order integrators such as the fourth order Runge-Kutta are also available.

1.2 Classification and Contribution

This thesis is concerned with research from the area of flow visualization. Four different approaches are widely used in flow visualization [116]: direct, geometric, texture-based, and feature-based flow visualization. These are illustrated in Figures 1.4 and 1.5 and explained in more detail below.

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. Intuitive pictures can be provided, especially in the case of two dimensions. Direct flow visualization approaches are likely amongst the oldest and well known. The use of color mapping in glyphs is standard in graphics software. Hence, this is not an area of focus in this thesis. However, we do make use of direct flow visualization in the context of a resampling approach as detailed in Chapter 3 [78], as well as in other cases. As such, direct flow visualization is a closely related topic.

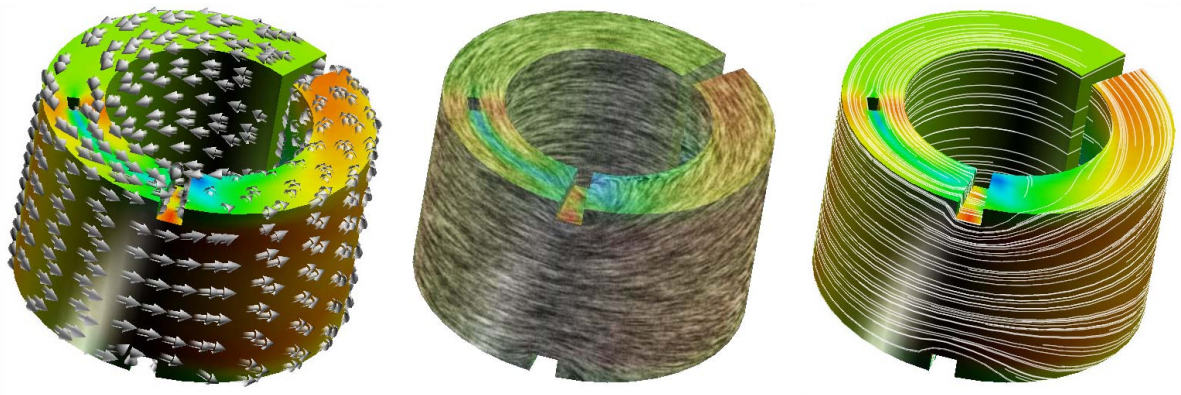


Figure 1.5: An example of circular flow at the surface of a ring to help illustrate our flow visualization classification: (left) direct visualization by the use of arrow glyphs, (middle) texture-based by the use of LIC, and (right) visualization based on geometric objects, here streamlines.

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 to generate isosurfaces, e.g., with respect to an isovalue of pressure or magnitude of velocity.

Geometric flow visualization techniques are one topic of research within this thesis. We apply a variety of techniques to CFD simulation data including oriented streamlines, dashed-animated streamlines, streamlets, and streamcomets. We also discuss an interactive technique that addresses the perceptual problems associated with flow visualization in 3D, namely a streamrunner [77].

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 moves is incorporated 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. In this thesis we use advection approaches according to Image Based Flow Visualization (IBFV) [164] and Image Space Advection (ISA) [83], which can generate both Spot Noise [163] and LIC-like [17] imagery. Both approaches are related to Lagrangian-Eulerian Advection (LEA) [64]. A full comparison of texture-based flow visualization techniques is given in the next chapter, the state-of-the-art. We focus on interactive visualization techniques because an interactive exploration of parameter space is essential for improving the design of automotive components that undergo CFD analysis.

The long computation time associated with texture-based approaches has been a problem since the introduction of these techniques themselves starting in the early 1990s [17, 163]. The computation time barrier was hurdled with the introduction of LEA [63] and again with IBFV [164]. IBFV solved the computation time problem for the case of 2D, unsteady flow. Most of the research work in this thesis hinges on the methodology necessary to extend texture-based techniques to surfaces in 3D for both steady and unsteady flow. With the introduction of ISA (Chapter 4 [83]) and IBFVS (IBFV for Curved Surfaces) [165] we saw texture-based flow visualization on surfaces at fast frame rates for the first time. And since these methods were introduced to the visualization community at the same time, a comparison of the two approaches was a natural choice for further research [85].

Feature-based flow visualization: For the sake of completeness, we note that feature-based flow visualization, another class of techniques including feature extraction and tracking, is beyond the scope of this thesis. The research we've done with Post et al. [116] covers feature-based flow visualization in detail. See Doleisch et al. [35, 36] for even more recent research results in this area.

Note that there are different amounts of computation associated with each category, as illustrated in Figure 1.4. In general, direct flow visualization techniques require less computation than the other three categories, whereas feature-based techniques require the most computation. This thesis focuses on the body of research related to geometric, dense, and texture-based techniques.

1.3 Overview

The rest of this thesis is organized as follows: In Chapter 2 we present the state-of-the-art techniques in geometric and texture-based flow visualization [82, 116]. A detailed overview of related research in the field is given. This also serves as an introduction to the main body of this thesis.

In Chapter 3 we present a practical approach to resampling of CFD simulation data from unstructured, adaptive resolution grids onto regular, rectilinear grids [78]. The technique also handles the case of unsteady flow. Chapter 4 presents a novel technique for the generation of dense, texture-based flow visualization on surfaces for the cases of both steady and unsteady flow [85]. The technique is named ISA [83] for Image Space Advection [85]. Chapter 5 applies ISA to the case of isosurfaces [84] and presents an argument with respect to the utility of the approach. Chapter 6 we discuss geometric flow visualization techniques, again, applied to CFD simulation data. Chapter 7 puts all these elements together and evaluates them by applying direct, geometric, and texture-based flow visualization techniques in order to investigate two important, common patterns of flow motion in CFD, namely the cases of swirl and tumble flow.

Chapter 8 gives some details about the design and implementation of the software modules used to build the flow visualization subsystem software. Some user interface concerns are also addressed. Finally, Chapter 9 presents a summary of all the work in the thesis and the final chapter presents some concluding remarks. Some interesting, related background material may also be found in the appendices.

Chapter 2

The State of the Art in Flow Visualization: Geometric, Dense, and Texture-Based Techniques

"The schoolteacher asks Billy Bob: 'If you have twelve sheep and one jumps over the fence, how many sheep do you have left?'
Billy Bob answers, 'None.'
'Well,' says the teacher, 'you sure don't know your subtraction.' 'Maybe not,' Billy Bob replies, 'but I darn sure know my sheep."'

– an old Texas joke

Flow visualization (FlowVis) is one of the classic subfields of visualization, covering a rich variety of applications, from the automotive industry, aerodynamics, turbomachinery design, to weather simulation, meteorology, climate modeling, ground water flow, and medical visualization. Consequently, the spectrum of FlowVis solutions is very rich, spanning multiple technical challenges: 2D vs. 3D solutions and techniques for steady or time-dependent data.

Bringing many of those solutions in linear order (as necessary for a text like this) is neither easy nor intuitive. Several options of subdividing this broad field of literature are possible. Hesselink et al., for example, addressed the problem of how to categorize techniques in their 1994 overview of research issues [55] and consider dimensionality as a means to classify the literature. In the following, several aspects are discussed on an abstract level before literature is addressed directly.

The classification we use here is described in Section 1.2. Figure 1.4 illustrates a classification of the aforementioned classes and Figure 1.5 shows three typical examples. The two main sections of this chapter are the texture-based flow visualization techniques in Section 2.5 and the geometric flow visualization techniques in Section 2.6. We also note the bulk of the material in this chapter has also be published elsewhere [82, 116, 117].

2.1 Spatial, Temporal, and Data Dimensionality

Solutions in flow visualization differ with respect to the dimensionality of the flow data. Useful techniques for 2D flow data, like color coding or arrow plots, sometimes lack similar advantages in 3D. Here, the spatial dimensionality of the flow data is indicated as either 2D, 2.5D (surfaces in 3D), or 3D. In our

classification the dimensionality of the results from each technique is marked with a corresponding label indicating dimensionality (see Figure 2.2).

By 2.5D we mean flow visualization restricted to surfaces in 3D. We draw attention to the notion that in many cases like CFD, the simulation sets all velocities on a surface to zero. One way to approach this is to extrapolate the vector field just inside the surface to the boundary. In any case, the vector component normal to the surface is usually lost in the visualization. Furthermore another vector field can be calculated on a surface, such as skin friction.

In addition to *spatial* dimensions, *temporal* dimension (dimensionality with respect to time) is of great importance. Firstly, velocity itself incorporates a notion of time – flows are often interpreted as differential data with respect to time (cf. Equation 2.1), i.e., when integrating the data, instantaneous paths such as streamlines may be obtained (cf. Equation 2.3). We call this *steady velocity time*. Additionally, the flow data itself can change over time resulting in time-dependent (or unsteady) flow. We refer to this as *unsteady velocity time*. The visualization must carefully distinguish between both. Performing integration in the case of unsteady data results in pathlines or streaklines as opposed to streamlines.

The distinction between steady and unsteady velocity time is important especially when animation is used in the visualization. Then, even a third notion of time, i.e., animation time, may affect the visualization. Animation time can be an arbitrary feature added to the visualization in order to create motion. Sometimes, geometric objects like streamlines are animated in order to show flow orientation, e.g., the motion of color controlled by a color-table [66]. Animation is also often added to texture-based methods with the same goal in mind. Special attention is required for correct interpretation of animation time.

In many cases, further *data* dimensions, i.e., attributes, are supplied with the data, such as temperature, pressure, or vorticity in addition to spatial and temporal dimensions. The dimension of the data values is also associated with the terms multivariate and multi-field data. Flow visualization may also take these values into account, e.g., by using color or isosurface extraction.

Although we do not have space to focus on experimental flow visualization, it is interesting to recognize that many computational solutions more or less mimic the visual appearance of well-accepted techniques in experimental visualization (cf. particle traces, dye injection, or Schlieren techniques [162]).

2.2 Data Sources

Computational flow visualization, in general, deals with data that exhibits temporal dynamics like the results from (a) *flow simulation* (e.g., the simulation of fluid flow through a turbine), (b) *flow measurements* (possibly acquired through laser-based technology), or (c) *analytic models* of flows (e.g., dynamical systems [1], given as set of differential equations).

We focus on visualization of data from computational flow simulation, i.e., flow data given as a set of samples on a grid. In many cases, the velocity information in a flow dataset (encoded as a set of velocity vectors) represents the focus. Therefore, flow visualization is strongly related to *vector field visualization*, which may also deal with vector fields other than velocity fields.

The relation of computational and experimental visualization is worthy of mention. Experimental flow visualization, as in a wind tunnel, is also used to validate computational flow simulation. In such a case the computational visualization needs to be set up in a way such that results can be easily compared.

2.3 Fundamentals

Before outlining some of the most important techniques, a short overview of common mathematics as well as some general concepts with regard to the computation of results are presented.

Flow simulations are often solutions to systems of PDEs, such as the Navier Stokes, Euler, or Advection-Diffusion equations [170]. In general, discretized solution methods are used. Noteworthy are finite volume (FV) and finite element (FE) analysis, which subdivide the domain into small elements like hexahedral or tetrahedral cells. A solution is defined on the computation grid in physical space: unstructured for FE and structured curvilinear for FV solutions. In the discussion that follows, we assume that vector data are defined on the grid nodes (cell vertices), although in some cases the vector data are defined at the cell centers.

Reconstruction of Flow Data

An inherent characteristic of flow data is that derivative information is given with respect to time and is laid out with respect to an n-dimensional spatial domain $\Omega \subseteq R^n$, e.g., $n = 3$ for representing 3D fluid flow. Recall, temporal derivatives $\mathbf{v}$ of nD locations $\mathbf{p}$ within the flow domain Ω are n-dimensional vectors:

$$\mathbf{v} = \mathrm{d}\,\mathbf{p} \,/\, \mathrm{d}\,t, \quad \mathbf{p} \in \Omega \subseteq R^n,\ \mathbf{v} \in R^n,\ t \in R \tag{2.1}$$

A general formulation of (possibly unsteady) flow data $\mathbf{v}$ is

$$\mathbf{v}(\mathbf{p}, t) : \Omega \times \Pi \rightarrow R^n \tag{2.2}$$

where $\mathbf{p} \in \Omega \subseteq R^n$ represents the spatial reference of the flow data and $t \in \Pi \subseteq R$ represents the system time. For steady flow data, the simpler case of $\mathbf{v}(\mathbf{p}) : \Omega \rightarrow R^n$ is given ($\mathbf{v}$ not dependent on t).

In results from nD flow simulation, such as from automotive applications or airplane design, vector data $\mathbf{v}$ is usually not given in analytic form, but requires reconstruction from the discrete simulation output. The numerical methods used for the flow simulation, such as finite element methods usually output simulation values on large-sized grids of many sample vectors $\mathbf{v}_i$, which discretely represent the solution of the simulation process. Furthermore, it is assumed that the flow simulation is based on a continuous model of the flow thus allowing continuous reconstruction of the flow data $\mathbf{v}$. One option is to apply a reconstruction filter $h : R^n \rightarrow R$ to compute $\mathbf{v}(\mathbf{p}) = \sum_i h(\mathbf{p} - \mathbf{p}_i)\,\mathbf{v}_i$. For practical reasons, filter h usually has only local extent. Efficient procedures for finding flow samples $\mathbf{v}_i$, which are nearest to the query point $\mathbf{p}$, are needed to do proper reconstruction.

Grids

In flow simulation, the vector samples $\mathbf{v}_i$ often are laid out across the flow domain with respect to a certain type of grid. Grid types range from simple rectilinear or Cartesian grids to curvilinear grids to complex unstructured grids (cf. Figure 2.1). Typically, simulation grids exhibit large variations in cell sizes. This variety of cell sizes stems from the influence of grid generation onto the flow simulation process. The quality of the grid model and its implementation impact the quality of the simulation results.

Although the principal theory of reconstruction from discrete samples does not exhibit many differences with respect to grid cell types, the practical handling does. While neighbor searching might be trivial in a rectilinear grid, it usually is not in a tetrahedral grid. Similar differences hold for the problems of point location and vector reconstruction. In the following we shortly describe some fundamental operations which form the basis for visualization computations on simulation grids.

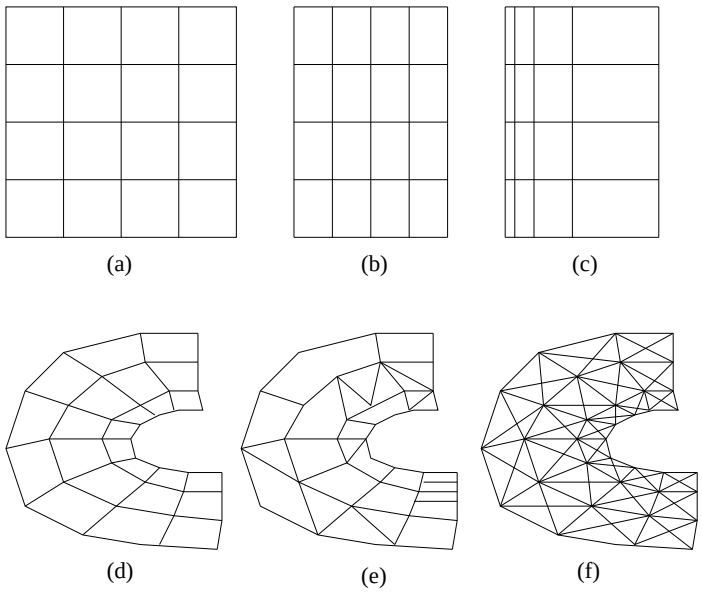


Figure 2.1: Grids involved in flow simulation – (a) Cartesian, (b) regular, (c) general rectilinear, (d) structured or curvilinear, (c) unstructured, and (f) unstructured triangular [78, 185].

Starting with *point location*, i.e., the problem of finding the grid cell in which a given nD-point lies, usually two cases are distinguished. For general point location, special data structures can be used that subdivide the spatial domain to speed up the search. For iterative point location, often needed during integral curve computation, algorithms are used that efficiently exploit spatial coherence during the search. One kind of such algorithms starts with an initial guess for the target cell, checks for point-containment and refines accordingly afterward. This process is iterated until the target cell is found. More details can be found in text books about flow visualization fundamentals [114, 142].

Beside point location, *flow reconstruction*, or interpolation, within a cell of the dataset is a crucial issue. Often, once the cell containing the query location is found, only the sample vectors at the cell's vertices are considered for reconstruction. The approach used most often is first-order reconstruction by performing linear interpolation within the cell. For example, trilinear flow reconstruction may be used within a 3D hexahedral cell.

After point location and flow reconstruction, visualization begins: vectors can be represented with glyphs, virtual particles can be injected and traced across the flow domain. Nevertheless, the *computation of derived data* may be necessary to do more sophisticated flow visualization. Often, the first step is to request second-order gradient information for arbitrary points in the flow domain, i.e., $\nabla \mathbf{v}|_{\mathbf{p}}$, which gives information about local properties of the flow (at point $\mathbf{p}$) such as flow convergence and divergence, or flow rotation and shear. For feature extraction, flow vorticity $\omega = \nabla \times \mathbf{v}$ can be of high interest. Further details about local flow properties can be found in previous work [96, 115].

Integration

Recalling that flow data in most cases is derivative information with respect to time, the idea of integrating flow data over time is natural to provide an intuitive notion of evolution induced by the flow. One example is visualization by the use of particle advection. A respective particle path $\mathbf{p}(t)$ – here through

unsteady flow – has been defined by

$$\mathbf{p}(t) = \mathbf{p}_0 + \int_{\tau=0}^{t} \mathbf{v}(\mathbf{p}(\tau),\, \tau)\, \mathrm{d}\tau \tag{2.3}$$

where $\mathbf{p}_0$ represents the location of the particle path at seed time 0. Note that Equations 2.1 and 2.3 are complimentary to each other. For other types of integral curves, such as streaklines, see previous work [76, 142].

As discussed in Chapter 1, it is important to note that respective integral equations like Equation 2.3 usually cannot be resolved for the curve function analytically, and thereby *numerical integration methods* are employed. The most simple approach is to use a first-order Euler method to compute an approximation $\mathbf{p}_E(t)$ – one iteration of the curve integration is specified by

$$\mathbf{p}_E(t + \Delta\, t) = \mathbf{p}(t) + \Delta\, t \cdot \mathbf{v}(\mathbf{p}(t),\, t) \tag{2.4}$$

where $\Delta\, t$ is often a very small step in time and $\mathbf{p}(t)$ denotes the location to start this Euler step from.

A more accurate but also more costly technique is the second-order Runge-Kutta method, [22, 118] which uses the Euler approximation $\mathbf{p}_E$ as a look-ahead to compute a better approximation $\mathbf{p}_{RK2}(t)$ of the integral curve:

$$\begin{aligned} &\mathbf{p}_{RK2}(t + \Delta\, t) = \\ &\quad \mathbf{p}(t) + \Delta\, t \cdot (\mathbf{v}(\mathbf{p}(t),\, t) + \mathbf{v}(\mathbf{p}_E(t + \Delta\, t),\, t))/2 \end{aligned} \tag{2.5}$$

Higher-order methods like the often used fourth-order Runge-Kutta integrator utilize more such steps to better approximate the local behavior of the integral curve. Also, adaptive step sizes are used to compute smaller steps in regions of high curvature.

2.4 Direct Flow Visualization

Direct, or global, flow visualization techniques attempt to present the complete data set, or a large subset of it, at a low level of abstraction. The mapping of the data to a visual representation is straightforward, without complex conversion or extraction steps. These techniques are perhaps the most intuitive visualization strategies as they present the data as is. Difficulties arise, when the long-term behaviour induced by flow data is investigated if direct FlowVis is used–this may require cognitive integration of visualization results.

Direct FlowVis in 2D

In this subsection we shortly address widely distributed, standard techniques for 2D FlowVis, i.e., coloring and arrow plots.

Color Coding in 2D – A common direct flow visualization technique is to map flow attributes such as velocity, pressure, or temperature to color. Since color plots are widely distributed, this approach results in very intuitive depictions. Of course, the color scale which is used for mapping must be chosen carefully with respect to perceptual differentiation. Color coding for 2D FlowVis extends to time-dependent data very well, resulting in moving color plots according to changes of the flow properties over time.

Arrow Plots in 2D – A natural vector visualization technique is to map a line, arrow, or glyph to each sample point in the field, oriented according to the flow field. Usually a regular placement of arrows is used in 2D, for example, on an evenly-spaced Cartesian grid. Two variants of arrow plots are often used: (1) normalized arrows of unit length which visualize the direction of the flow only and

(2) arrows of varying length that is proportional to the flow velocity. Klassen and Harrington [74] and Schroeder et al. [135] call this technique a hedgehog visualization because of the bristly result. Two dimensional hedgehog plots can be extended to time-dependent data, although bigger time steps might result in jumping arrows, diminishing the quality of such a visualization.

Hybrid direct FlowViz in 2D – Kirby et al. propose simultaneous visualization of multiple values (of 2D flow data) by using a layering concept related to the painting process of artists [72]. Arrow plots are mixed with color coding to provide visualization results rich of information.

Direct FlowViz on Slices or Boundaries – When dealing with 3D flow data, visualization naturally faces additional challenges such as 3D rendering. Acting as a middle ground between 2D FlowViz and the visualization of truly 3D flow data is the restriction to subdimensional parts of the 3D domain, e.g., sectional slices or boundary surfaces. Thereby, techniques known from 2D FlowViz often are applicable without major changes, at least from a technical point of view. When working with sectional slices, the treatment of flow components orthogonal to slices requires some special care.

Color Coding on Slices or Boundaries – Color coding is very effective for visualizing boundary flows or sectional subsets of 3D flow data. A good example is NASA s Field Encapsulation Library [106], which allows to easily use both techniques for various flow data. Schulz et al. also use color coding of scalars on 2D slices in 3D automotive simulation data [137]. They introduce an interactive slicing probe which maps the vector field data to hue. The use of scalar clipping, i.e., the transparent rendering of slice regions where the corresponding data value does not lie within a specific data range, allows to use multiple (colored) slices with reduced problems due to occlusion.

2D Arrows on Slices or Boundary Surfaces – Using 2D arrows on slices from 3D flow data is also an effective visualization technique [42]. However, results of such a visualization should be interpreted carefully, as flow components which are orthogonal to the slice are usually not depicted. The above mentioned difficulties with 2D arrows and sectional slices through 3D flow are basically negligible, when talking about boundary surfaces, since in these cases, rarely cross-boundary flows are given. Therefore the use of arrows spread out over boundary surfaces usually is very effective, as used by Treinish for weather visualization [157].

Direct FlowVis in 3D

After discussing direct FlowVis on slices and boundary surfaces, direct FlowVis of real 3D flow is discussed. In contrast to previously mentioned techniques, here rendering becomes the most critical issue. Occlusion and complexity make it difficult (if possible at all) to get an immediate overview of an entire flow data set in 3D.

Volume Rendering for 3D FlowViz – The natural extension of color coding in 2D (or on slices, etc.) is color coding in 3D. This, however, poses special requirements onto rendering due to occlusion problems and nontrivial complexity–volume rendering is needed. Volume rendering is well-known in the field of medical 3D visualization, i.e., volume visualization. However, those challenges, which closely correspond to flow visualization are briefly addressed here: (1) flow data sets are often significantly smoother than medical data–an absence of sharp and clear object boundaries (like organ boundaries) makes mapping to opacities more difficult and less intuitive. (2) flow data is often given on non-Cartesian grids, e.g., on curvilinear grids the complexity of volume rendering gets significantly more difficult, starting with nontrivial solutions required for visibility sorting and blending. (3) flow data is also time-dependent in many cases, imposing additional loads on the rendering process.

In the early nineties, Crawfis et al. [25, 24], as well as Ebert et al. [39] applied volume rendering techniques to vector fields. Later, Frühauf applied ray casting to vector fields [46]. Recently, Westermann,

presented a relatively fast 3D volume rendering method using a resampling technique for vector field data from unstructured to Cartesian grids [182].

Recently, Clyne and Dennis [21] as well as Glau [48] presented volume rendering for time-varying vector fields using algorithms which make special use of graphics hardware. Ono et al. use direct volume rendering to visualize thermal flows in the passenger compartment of an automobile [111]. Their goal is to attain the ability to predict the thermal characteristics of the automotive cabin through simulation. Swan et al. apply direct volume rendering techniques in flow visualization in a system that supports computational steering [146]. Their visualization results are extended to the CAVE environment. Recently, Ebert and Rheingans demonstrated the use of nonphotorealistic volume rendering techniques for 3D flow data [37]. They apply silhouette enhancement or tone shading to improve renderings of 3D flows.

Arrow Plots in 3D – The use of arrows for direct 3D FlowVis poses at least two problems: (1) the position and orientation of a vector is often difficult to understand because of its projection onto a 2D screen–using 3D representations of arrows (like a cylinder plus a cone) decreases these problems with perception and (2) glyphs occluding one another become a problem. Careful seeding is required (in contrast to the default of dense distributions). In actual applications, arrow plots are usually based on selective seeding, for example, all arrows starting from one out of a few sectional slices through the 3D flow.

Boring and Pang address the problem of clutter in 3D direct FlowVis by highlighting those parts of a 3D arrow plot, which point in a similar direction compared to a user-defined direction [11]. Their methodology reduces the amount of data being displayed thus results in less clutter. Their methods can be combined with other techniques that use glyph representations and flow geometries such as streamlines for FlowVis. They apply the methods to both analytic and simulation data sets to highlight flow reversals.

2.5 Dense and Texture-Based Flow Visualization

Dense, texture-based techniques in flow visualization generally provide full spatial coverage of the vector field. In our classification we group these methods into the following categories based on their respective *primitive*: the fundamental object upon which the algorithm is based. Our classification subdivides the techniques based on their similarity.

- *Spot Noise techniques:* These methods (Section 2.5) are based on a technique introduced by Van Wijk [163]. In this category, the basic primitive on which the algorithms operate is the so-called *spot*: an ellipse or other shape that is warped and distributed in order to reflect the characteristics of a vector field.
- *LIC techniques:* The methods in this category (Section 2.5) are derived from an algorithm introduced by Cabral and Leedom [17], namely, Line Integral Convolution (LIC). The basic primitive here is a *noise texture*: the properties of the texture are convolved, or smeared, using a kernel filter in the direction of the underlying vector field.
- *Texture advection and GPU-based techniques:* The primitive in this case (Section 2.5) is a *moving texel* [105]. Individual texels/texel properties, or groups of texels are advected in the direction of the vector field. Many of the techniques in this category utilize more computation on the GPU (Graphics Processing Unit) – rather than the CPU – in order to realize performance gains.
- *Related techniques:* Most of the dense, texture-based flow visualization research falls into one of the previous categories. Related research that does not fit cleanly into one of the previous classifications is discussed in Section 2.5.

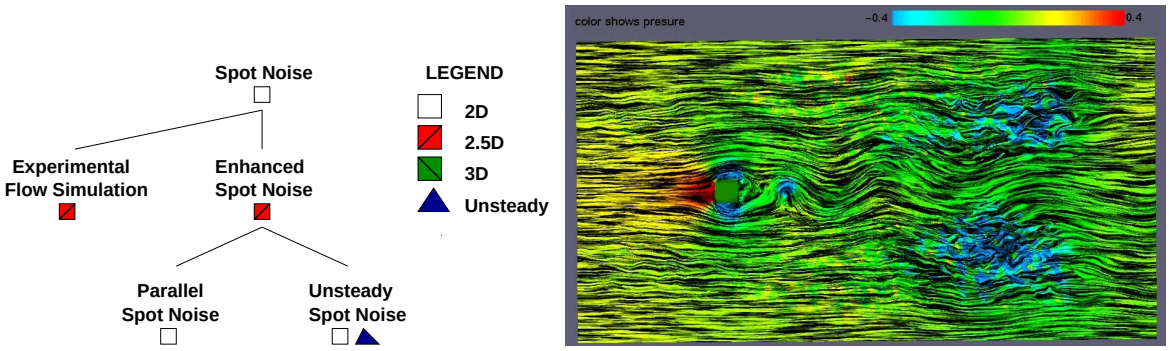


Figure 2.2: (left) The Spot Noise hierarchy of related research. Children in the hierarchy build upon the work of their parent. (right) A snapshot of the unsteady spot noise algorithm [31]. Image courtesy of De Leeuw and Van Liere.

We have included a section of meta-research papers in Section 2.7 after the individual research techniques. These papers attempt to provide an alternative, higher-level framework that incorporates several of the techniques discussed here.

Spot Noise

Spot noise, introduced by Van Wijk [163], was one of the first dense, texture-based techniques for vector field visualization. Spot noise generates a texture by distributing a set of intensity functions, or spots, over the domain. Each spot represents a particle warped over a small step in time and results in a streak in the direction of the local flow from where the particle is seeded. A spot noise texture is defined by [163]:

$$f(\mathbf{x}) = \sum a_i h(\mathbf{x} - \mathbf{x}_i, \mathbf{v}(\mathbf{x}_i)) \tag{2.6}$$

in which $h()$ is called the intensity function, a_i is a scaling factor, and $\mathbf{x}_i$ is a random position. A spot is a function with unity intensity value for the spot, e.g., a ellipse and its interior, and zero everywhere else. The summation denotes the blending of each instance of the intensity function at random positions.

The hierarchy shown in Figure 2.2, left illustrates the relationship amongst spot noise related methods. Follow-up research that builds upon a previous technique is shown as a child in the hierarchy. Children that share a common parent are presented in chronological order of appearance when reading from left to right. Each node in the hierarchy is labeled and the corresponding description can be matched in the text of this chapter. The dimensionality of the flow data used to generate the results is indicated for convenience. The time dimension label is given a different shape to distinguish it from the spatial dimensions. We believe the spot noise hierarchy (Figure 2.2) and the LIC hierarchy (Figure 2.4) will be valuable assets in helping the reader navigate the related research literature. In what follows, we visit each node in the hierarchy in depth-first-search order.

Comparative Visualization – Spot noise has been used to simulate the results from the field of experimental flow visualization [29]. First the parameters of the spot noise technique are tuned in order to simulate the smearing of oil on a surface. A post-processing step is then added to enhance the visualization result such that it looks closer to the smearing of real oil from experimental flow visualization.

Enhanced Spot Noise – One limitation of the original spot noise algorithm was the inability to represent high, local velocity curvature especially with high speeds. Enhanced spot noise [33] by De Leeuw and Van Wijk addresses these challenges through the use of bent spot primitives.

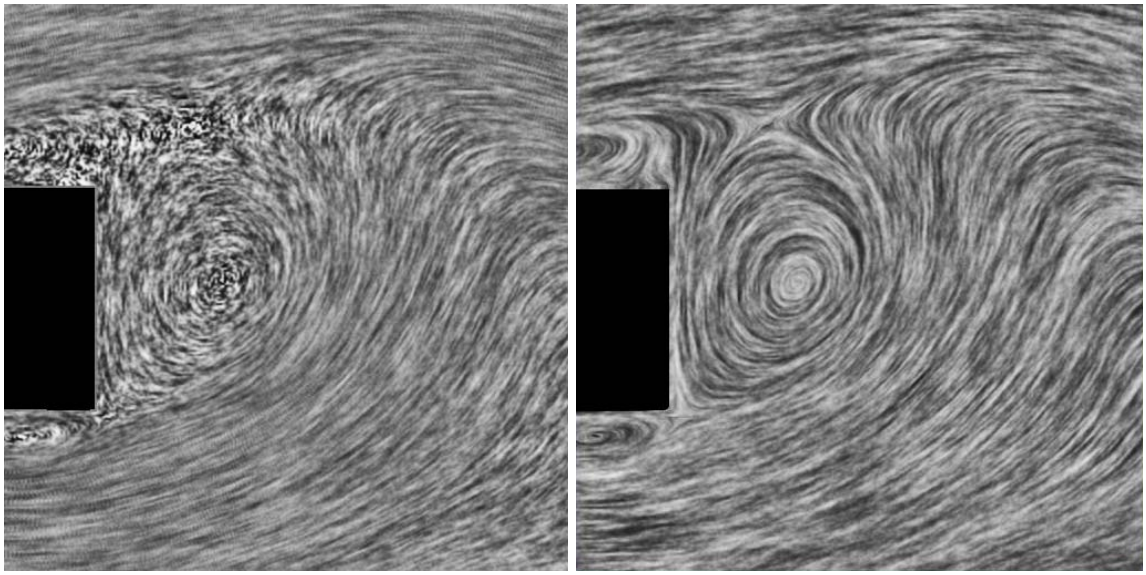


Figure 2.3: Visualization of flow past a box using (left) spot noise and (right) LIC.

Parallel and Unsteady Spot Noise – In order to accelerate the performance of enhanced spot noise towards interactive frame rates, a parallel implementation of the algorithm was introduced by De Leeuw [28]. The parallel implementation was applied to the steering of a smog prediction simulation and searching a very large data set resulting from a numerical simulation of turbulence.

The first application of spot noise to unsteady flow is presented by De Leeuw and Van Liere [31] (Figure 2.2 right). The motion of spots is modeled after particles in unsteady flow. In order to visualize unsteady flow, the distribution of spots with respect to the temporal domain is discussed. Unsteady spot noise also introduces support for zooming views of the vector field. Spot noise with zooming is also utilized by De Leeuw and Van Liere when visualizing topological features of 2D flow [26].

Spot Noise Related Literature – A combination of both texture-based FlowVis on 2D slices and 3D arrows for 3D flow visualization is employed by Telea and Van Wijk [154]. Arrows denote the main characteristics of the 3D flow after clustering and a 2D slice with spot noise visualization serves as context. The focus of this work is on vector field clustering.

Löffelmann et al [95] use anisotropic spot noise created from a grid-shaped spot to visualize streamlines and timelines concurrently on stream surfaces. Another interesting application of spot noise is its use for the depiction of discrete maps (non-continuous flow) [93].

Spot noise has also been applied to the visualization of turbulent flow [30] and in combination with the visualization of flow topology [26, 27]. We refer the reader to Post et al. [116, 117] for more on the subject of flow topology.

Spot Noise vs. LIC – A visual comparison of LIC (the focus of the next section) and spot noise is shown in Figure 2.3. Spot noise is capable of reflecting velocity magnitude within the amount of smearing in the texture, thus freeing up hue for the visualization of another attribute such as pressure, temperature, etc. On the other hand, LIC is more suited for the visualization of critical points which is a key element in conveying the flow topology. The vector magnitudes are normalized thus retaining lower spatial frequency texture in areas of low velocity magnitude. De Leeuw and Van Liere also compare spot noise to LIC [32]. They report that LIC is better at showing direction than spot noise, but it does

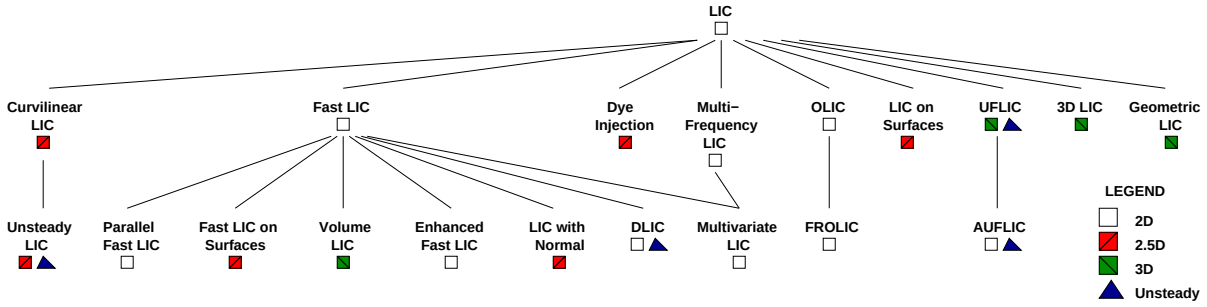


Figure 2.4: The LIC hierarchy of related research. Node labels correspond to paragraphs in the text, which then lead to specific entries in the bibliography.

not encode velocity magnitude. By flow direction, we refer to the path along which a massless particle follows when injected into the flow.

Line Integral Convolution

Line integral convolution (LIC), first introduced by Cabral and Leedom [17], has spawned a large collection of research as indicated in Figure 2.4. The original LIC method takes as input a vector field on a 2D, Cartesian grid and a white noise texture of the same size. Texels are *convolved* (or correlated) along the path of streamlines using a *filter kernel* in order to create a dense visualization of the flow field. More specifically, given a streamline $\boldsymbol{\sigma}$, LIC consists of calculating the intensity I for a pixel located at $\mathbf{x}_0 = \sigma(s_0)$ by [144]:

$$I(\mathbf{x}_0) = \int_{s_0-L/2}^{s_0+L/2} k(s - s_0) T(\sigma(s))\, ds \tag{2.7}$$

where T stands for an input noise texture, k denotes the (usually symmetric) filter kernel, s is an arc length used to parameterize the streamline curve σ, and L represents the filter kernel length. See Figure 1.5 (middle) for a result. LIC was one of the first dense, texture-based algorithms able to accurately reflect velocity fields with high local curvature.

The research in LIC-based flow visualization described here extends LIC in several directions: (1) adding flow orientation cues, (2) showing local velocity magnitude, (3) adding support for non-rectilinear grids, (4) animating the resulting textures such that the animation shows the upstream and downstream flow direction, (5) allowing real-time and interactive exploration, (6) extending LIC to 3D, and (7) extending LIC to unsteady vector fields. In the following, we visit the LIC hierarchy of Figure 2.4 in depth-first-search order.

Curvilinear Grids and Unsteady LIC – Forssell [43] was early to extend LIC to surfaces represented by curvilinear grids. The original LIC method portrays a vector field with uniform velocity magnitude. Forssell introduces a technique for displaying vector magnitude. She also describes one approach to animate the resulting LIC textures. Forssell and Cohen extend this work to visualize unsteady flow [44]. Their approach modifies the convolution such that the filter kernel operates on streaklines rather than streamlines. In other words, they modify the LIC algorithm to trace a path that incorporates multiple time steps.

Fast LIC – Many algorithms are built on Fast LIC introduced by Stalling and Hege [144]. Fast LIC is approximately one order of magnitude faster than the original. The speedup is achieved through two key observations: (1) fast LIC minimizes the computation of redundant streamlines present in the original

method and (2) fast LIC exploits similar convolution integrals along a single streamline and thus re-uses parts of the convolution computation from neighboring streamline texels. They also introduce support for filtered images at arbitrary resolution.

Parallel Fast LIC – Amongst the first parallel implementations of fast LIC is that of Zöckler et al. [187]. The algorithm computes animation sequences on a massively parallel distributed memory computer. Parallelization is performed in image space rather than in time in order to take advantage of the strong temporal coherence between frames. Luckily, as we shall see later, flow visualization research in this area has evolved far enough such that expensive parallel processing hardware is not always necessary to achieve interactive visualization [63, 64, 83, 164]. However, for 3D and unsteady flow there is still need for parallelization. For the sake of completeness, we also mention the work of Cabral and Leedom on parallelization of LIC [16] although this is a parallel processing version of the original LIC algorithm, not fast LIC.

Fast LIC on Surfaces – Battke et al. [3] extend fast LIC to surfaces represented by arbitrary grids in 3D. The approach by Forssell and Cohen [44] was limited to surfaces represented by curvilinear grids. The method works by tessellating a given surface representation with triangles. The triangles are packed (or tiled) into texture memory and a local LIC texture is computed for each triangle. The results presented here are limited to relatively small simple surface representations composed of equilateral triangles (1,600–4,000 triangles).

Volume LIC – Interrante and Grosch [60, 61] visualize true 3D flow using the fast LIC algorithm as a starting point. Clearly, there are perceptual challenges related to 3D flow visualization such as occlusion, depth perception, and visual complexity. Volume LIC introduces the use of *halos* in order to enhance depth perception such that the user has a better chance at perceiving the 3D space covered in the visualization (Figure 2.5). Areas of higher velocity magnitude are mapped to higher texture opacity. It is interesting to note that with the introduction of halos, we are then able to identify distinct entities in the 3D field, a property generally not present in other LIC techniques. Thus the 3D LIC takes a step in the direction of being a geometric flow visualization technique where discrete integral objects such as streamlines can be distinguished. Without introducing some notion of sparseness into the visualization, the results would not be very useful. However, with the introduction of sparseness, a trade-off is made between flow field coverage and reducing occlusion.

Enhanced Fast LIC and LIC with Normal – Two useful extensions to the fast LIC algorithm are introduced by Hege and Stalling [51] and Scheuermann et al. [132]. Hege and Stalling [51] experiment with higher order filter kernels in order to enhance the quality of the resulting LIC textures.

In the case of slices, vector components orthogonal to the slice are removed when using texture-based and geometric methods for visualization results. Scheuermann et al. [132] address this missing orthogonal vector field component by extending fast LIC to incorporate a normal component into the visualization.

DLIC – Sundquist [145] presents an extension to fast LIC, DLIC (Dynamic LIC), in order to visualize time-dependent electromagnetic fields. According to Sundquist, the motion of the field is not necessarily along the direction of the field itself in the case of electromagnetic fields. The algorithm proposed here handles the case of when the vector field and the direction of the motion of the field lines are independent. Conceptually, there are two vector fields used in this approach: (1) the electromagnetic field itself and (2) the vector field that describes the evolution of streamlines as a function of time.

Multivariate LIC – Urness et al. [161] present an extension to fast LIC that incorporates a new coloring scheme that can be used to incorporate multiple 2D scalar and vector attributes. *Color weaving* assigns a specific attribute represented by a color to an individual streamline thread in the visualization.

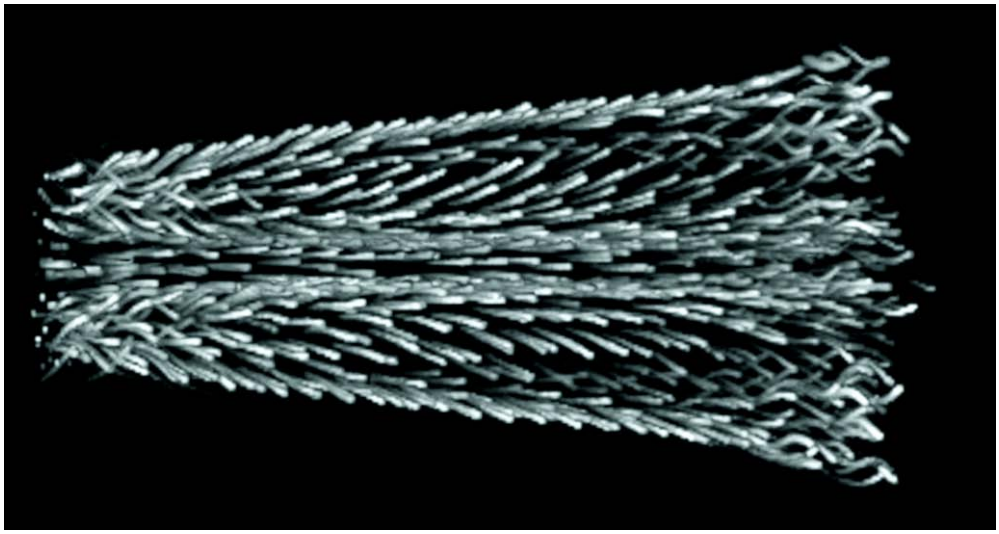


Figure 2.5: A result from the Volume LIC method [60, 61]. Image courtesy of Interrante and Grosch.

The streamline patterns may interweave and so may the color patterns. Using multiple colors allows visualization of more than one variate in the result. Their second contribution is called *texture stitching*: an extension to the idea presented by Kiu and Banks [73], namely multi-frequency LIC. However, in the case of Urness et al. [161] the multi-frequency noise textures are used to highlight regions of interest as opposed to velocity magnitude as by Kiu and Banks [73].

Dye Injection – Shen et al. address the problem of directional cues in LIC by incorporating animation and introducing dye advection into the computation [138]. The simulation of dye may be used to highlight features of the flow. In addition, they incorporate volume rendering methods that map a LIC texture onto a 3D surface. Thus the user is able to visualize the dye throughout the volume. We point out that the modeling of dye transport is not always physically correct since dye is dispersed not only by advection, but also by diffusion. Note that dye advection techniques can be classified differently. Dye injection can result in discrete geometric objects used to visualize the flow, and thus, could be classified as a group of geometric visualization techniques. Dye injection is also implemented by some of the texture advection and GPU-based techniques as described in Section 2.5.

Again, in Shen et al. we see the notion of a sparser visualization (as compared to LIC) in order to see into the 3D flow. The resulting 3D visualization approaches that of a geometric technique such as the use of streamsurfaces. And just as with the other geometric techniques, the notion of where to place or inject the dye into the flow becomes important.

Multi-Frequency LIC – Kiu and Banks propose to use a multi-frequency noise for LIC [73]. The spatial frequency of the noise is a function of the magnitude of the local velocity. Long, fat streaks indicate regions of the flow with higher velocity magnitude.

One problem with many curvilinear grid LIC algorithms is that the resulting LIC textures may be distorted after being mapped onto the geometric surfaces, since a curvilinear grid usually consists of cells of different sizes. Mao et al. propose a solution to the problem by using multi-granularity noise as the input image for LIC [100].

OLIC and FROLIC – Wegenkittl et al. address the problem of direction of flow in still images with their OLIC (Oriented LIC) approach [176]. By orientation, they mean the upstream and downstream

directions of the flow, not visible in the original LIC implementation. Conceptually, the OLIC algorithm makes use of a sparse texture consisting of many separated spots that are smeared in the direction of the local vector field through integration. A fast version of OLIC, called FROLIC (Fast Rendering of OLIC), is achieved by Wegenkittl and Gröller [175] via a trade-off of accuracy for time. FROLIC approximates the simulated droplet trace resulting from OLIC with a sequence of disks of varying intensity, with disk intensity increasing towards the downstream direction.

Animated FROLIC [7] achieves animation of the result via a color-table and is based on the observation that only the colors of the FROLIC disks need to be changed. Each pixel is assigned a color-table index that points to a specific entry in the color-table. Color-table animation then changes the entries of the color-table itself rather than the pixels of the corresponding image.

LIC on Surfaces – Mao et al. [101] extend the original LIC method by applying it to surfaces represented by arbitrary grids in 3D. Former LIC methods targeted at surfaces were restricted to structured grids [43, 44, 138]. Also, mapping a computed 2D LIC texture to a curvilinear grid may introduce distortions in the texture. Mao el al. propose solutions to overcome these limitations. The principle behind their algorithm relies on solid texturing [112]. The convolution of a 3D white noise image, with filter kernels defined along the local streamlines, is performed only at visible ray-surface intersections.

This idea has an advantage over that of Battke et al. [3] in that it avoids what can be a timely and complex assembly of triangles into texture space. However, ray-tracing is also costly. The method here is viewpoint dependent and required relatively lengthy processing time for an unstructured mesh composed of 10,000 triangles.

A significant body of research is dedicated to the extension of LIC onto boundary surfaces. Teitzel et al. [151] present an approach that works on both 2D unstructured grids and directly on triangulated grids in 3D space. This topic itself is the subject of a survey by Stalling [143].

UFLIC – Shen and Kao [139] extend the original LIC algorithm to handle unsteady flows. Their extension, called UFLIC (Unsteady Flow LIC), handles the case of unsteady flow fields by introducing a new convolution filter that better models the nature of unsteady flow. The convolution is done along pathlines (as opposed to streamlines). They improve upon the shortcomings of the previous unsteady LIC attempt presented by Forssell and Cohen [44]. According to Shen and Kao, Forssell and Cohen's approach has multiple limitations including: (1) lack of clarity with respect to spatial coherence, (2) deriving current flow values from future flow values, (3) unclear exposition with respect to temporal coherence, and (4) lack of accurate time stepping. All of these problems are addressed by UFLIC. Shen and Kao also apply UFLIC to the visualization of time-dependent flow to parameterized surfaces. UFLIC is also extended using a parallel implementation by Shen and Kao [140].

AUFLIC – AUFLIC (Accelerated UFLIC) is an extension to UFLIC that enhances performance times [89]. The principle behind AUFLIC is to save, re-use, and update pathlines in a vector field seeding strategy. AUFLIC requires approximately one half of the time required by UFLIC and generates similar results.

3D LIC – Rezk-Salama et al. [121] propose rendering methods to effectively display the results of 3D LIC computations. They utilize texture-based volume rendering in an effort to provide exploration of 3D LIC textures at interactive frame rates. Like Interrante [61], they address the perceptual problems posed by dense, 3D visualization. They approach these challenges through the use of opacity transfer functions and clipping planes, as in Figure 2.6. Opacity transfer functions allow the user to see through portions of the LIC textures deemed uninteresting by the user. In addition to conventional clipping planes, Rezk-Salama et al. also use clipping with arbitrary closed-surface geometries.

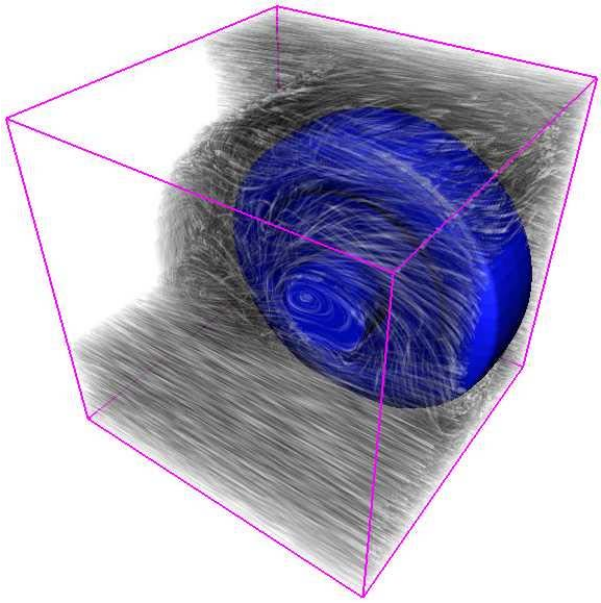


Figure 2.6: An LIC visualization showing a simulation of flow around a wheel [121]. The appropriate choice of transfer function results in a sparser noise texture. Image courtesy of Rezk-Salama et al. [121]

The use of transfer functions and geometric clipping objects are interesting choices for dealing with the perceptual problems associated with 3D. In some sense, these can be compared with the seeding problem of the geometric class of visualization techniques. Seeding strategies address where to start streamlines and other integration-based geometric objects. Selective seeding of geometric objects in 3D is often considered a key to successful visualization. However, knowledge of the proper seed locations is a requisite for this approach. And just as proper seed placement is a requisite when using geometric objects, expertise with the transfer function(s) and closed-object clipping geometries is required in the case of 3D LIC.

Geometric LIC – We make a distinction between geometric flow visualization and dense, texture-based flow visualization. However, these two topics are closely coupled. Conceptually, the path from using geometric objects to texture-based visualization is obtained via a dense seeding strategy. That is, densely seeded geometric objects result in an image similar to that obtained by dense, texture-based techniques [65]. Likewise, the path from dense, texture-based visualization to visualization using geometric objects is obtained using something such as a sparse texture for texture advection [176].

Here we have grouped together techniques that synthesize LIC results by mapping a pre-computed LIC texture onto geometric primitives such as streamlines. By using geometric primitives, researchers hope to speed up performance times of the LIC results. The drawback of these methods is that they require careful seeding strategies to gain the complete coverage of the flow field offered by traditional LIC techniques.

Motion Map – Jobard and Lefer use a *motion map* [66] in order to animate 2D steady flows. First, the domain is covered completely with streamlines. Next, a color is mapped to the streamlines and a color-table animation technique is used to animate the flow. It offers the advantage of saving memory and computation time since only one image of the flow has to be computed and stored in the motion map data structure. This technique is not applicable to unsteady flow however. It relies on a one-time cost of computing a set of streamlines.

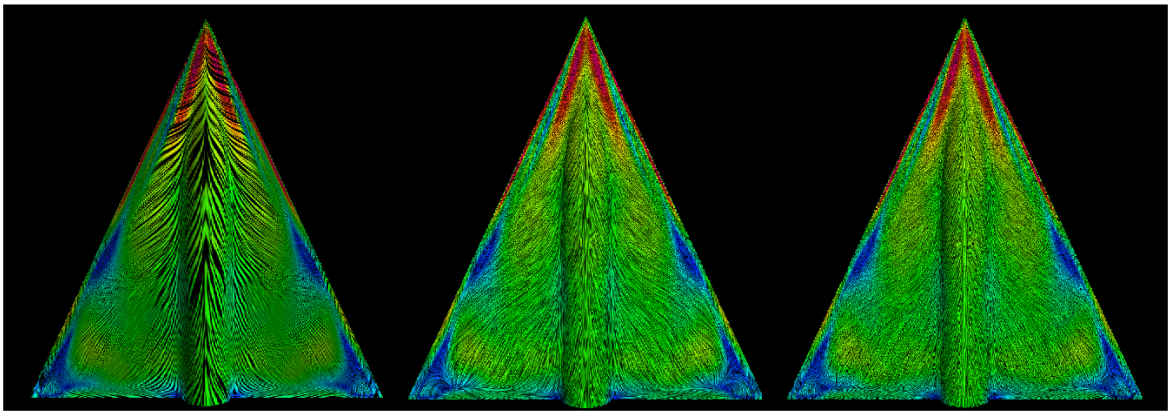


Figure 2.7: A comparison of 3 LIC techniques: (left) UFLIC [140], (middle) ELIC [110], and (right) PLIC [168]. Image courtesy of Verma et al. [168].

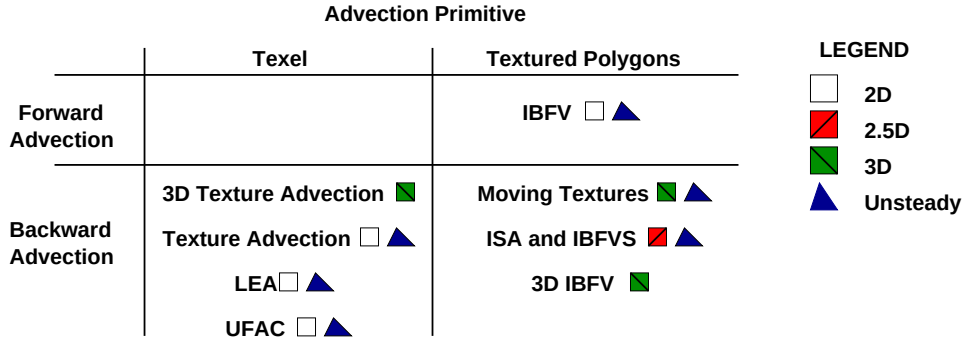


Figure 2.8: The classification of texture advection and GPU-based techniques. The columns indicate the primitive used during advection while the rows indicate the advection scheme.

PLIC – Verma et al. present a method for visual comparison of streamlines and LIC called PLIC [168] (Pseudo-LIC). They attempt to identify the relevant parameters to generate LIC-like images from a dense set of streamlines and for generating streamline-like images through the use of different filters used for convolution. By experimenting with different input textures for LIC, both streamline-like images and LIC-like results can be obtained. ELIC (enhanced LIC), placed here because of its visual comparison with PLIC, builds on the original LIC algorithm in four ways: (1) by incorporating an algorithm to improve the delineation of streamlines, (2) increasing the image contrast, (3) removing texture distortion introduced by applying LIC to curvilinear grids, and (4) using color to highlight flow separation and reattachment boundaries. A visual comparison between UFLIC [140], ELIC [110], and PLIC is shown in Figure 2.7.

Hierarchical LIC – Bordoloi and Shen [10] introduce a hierarchical approach to LIC based on a quadtree data structure used to support level of detail (LOD). The idea is to replace portions of the vector field of lower complexity with rectangular LIC texture-mapped patches. The LIC texture is taken from a previously calculated LIC image of a straight vector field. Here, complexity is a direct function of the amount of curl in the local vector field.

Decoupled LIC – Li et al. [88] present a technique for the visualization of 3D flow based on texture mapped primitives, namely streamlines. They decouple the visualization into a pre-processing type stage that computes the streamlines and a stage which maps various textures to the streamlines computed in the first stage. The result is volume rendered at interactive frame rates. To address the perceptual

challenges posed by 3D visualization, depth cues, lighting effects, silhouettes, shading, and interactive volume culling are described.

Texture Advection and GPU-Based Techniques

In this section we describe research based on moving texels or moving groups of texels, i.e., texture-mapped polygons whose motion is directed by the vector field. Figure 2.8 shows an overview of the different techniques and classifies them according to two properties: (1) the advection scheme used and (2) the primitive used during advection. Some of the literature focuses mainly on the integration scheme used to advect textures or texels. By the term texel we mean texture element. Some methods focus on the mapping to advected primitives and some focus on both. Figure 2.8 also shows the dimensionality of the flow data. In our discussion, we visit the methods in clockwise order starting at 12 o'clock. Within each sub-block the methods are listed in chronological order. This is because the mapping of texel properties between two time steps in the visualization is not 1-to-1 in this case. For a more detailed discussion see Jobard et al. [63, 64]

One characteristic common to many of the texture advection techniques in this section [63, 64, 83, 102] is the use of *backward coordinate integration* (or backward advection). None of the methods described here use forward advection (i.e., forward integration) and individual texels as a primitive. This is because the combination of forward integration and texel primitives leaves holes in the visual domain after the forward integration computation [64]. Given a position, $\mathbf{x}_0(i,j) = (i,j)$ of each particle in a 2D flow, backward integration over a time interval h determines its position at a previous time step [63]:

$$\mathbf{x}_{-h}(i,j) = \mathbf{x}_0(i,j) + \int_{\tau=0}^{h} \mathbf{v}_{-\tau}(\mathbf{x}_{-\tau}(i,j))d\tau \tag{2.8}$$

where h is the integration step, $\mathbf{x}_{-\tau}(i,j)$ represents intermediary positions along the pathline passing through $\mathbf{x}_0(i,j)$, and $\mathbf{v}_\tau$ is the vector field at time τ. We note that the methods in this category are generally implemented in an *iterative* fashion. That is for each animated frame an integration is performed over a small time-step h, followed by an update of visual properties. This is opposed to geometric methods in which a longer particle path may be computed over several time steps before the results are displayed.

IBFV – Image Based Flow Visualization (IBFV) by Van Wijk [164] is one of the fastest algorithms for dense, 2D, unsteady vector field representations (Figure 2.9). It is based on the advection and decay of textures in image space. Each frame of the visualization is defined as a blend between the previous image, warped according to the flow direction, and a number of background images composed of filtered white noise textures. One reason it is faster than many texture-based flow visualization methods is because it reduces the number of integration computations that need to be performed via advecting small quadrilaterals rather than individual pixels.

Moving Textures – Max and Becker were early to introduce the idea of moving textures in order to visualize vector fields [102]. One of the primary goals of this work was to use textures in motion to produce near-real-time animation of flow. Texture-mapped triangles are advected, or distorted, in the direction of the flow. Also, applying this technique to 3D flows with no modification provides results that are difficult to perceive, at least in the case of a still image.

ISA and IBFVS – IBFV has been extended to the visualization of flow on surfaces [83, 165]. Van Wijk presents an extension called IBFVS, IBFV for Surfaces [165]. Laramee et al. [83] present a similar dense, texture-based visualization technique on surfaces for unsteady flow called ISA: Image Space Advection. Both methods produce animated textures on arbitrary 3D triangle meshes in the same manner as the original IBFV method. Textures are generated, advected, and blended in image space. The methods

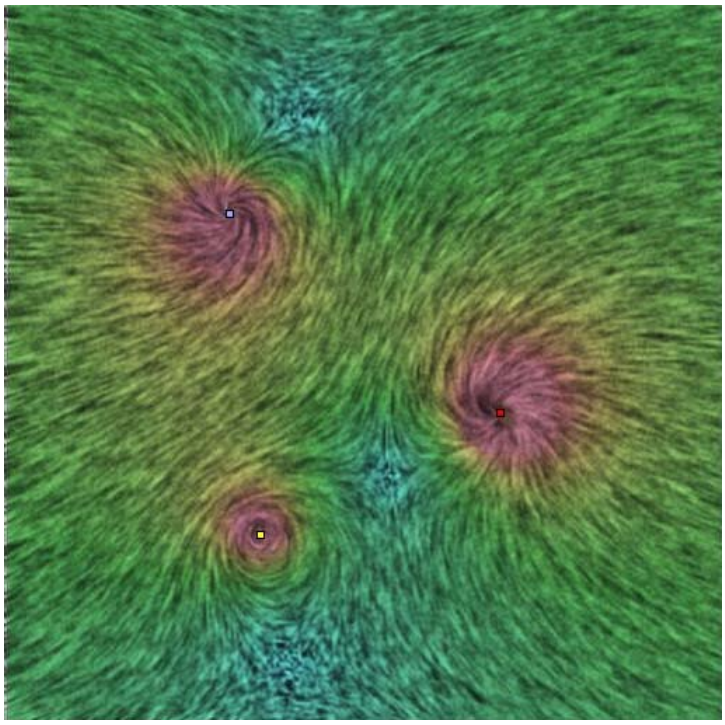


Figure 2.9: A screen shot from the Image Based Flow Visualization algorithm. Image courtesy of Van Wijk [164].

generate dense representations of time-dependent vector fields with high spatio-temporal correlation. While the 3D vector fields are associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space. Both spot noise and LIC-like results can be attained. In both techniques, [83, 165] fast frame rates are achieved in part by exploiting the GPU.

Van Wijk's method is applied to potential field visualization and surface visualization. Laramee et al.'s algorithm is applied to unsteady flow on boundary surfaces of large, complex meshes from computational fluid dynamics, dynamic meshes with time-dependent geometry and topology. It has also been applied to medical simulation data as well as isosurfaces [84]. A full comparison of the methods has been given by Laramee et al. [85].

3D IBFV – IBFV has also been applied to the visualization of 3D flow [153]. The problem of how to see inside the flow volume is addressed by varying both the noise sparsity, reminiscent of Interrante and Grosch [61], and through varying the opacity of the rendered volume similar to Rezk-Salama et al [121]. In order to achieve sparseness, Telea and Van Wijk inject empty holes of noise into the 3D field, in addition to the noise described by the original IBFV. One important component of their method is to define a threshold value which eliminates all close-to-transparent texel values. One disadvantage of the method is that the range of velocity values it can display is limited: A texel property cannot be advected by more than one slice along the z axis of the volume in one animation frame. This problem is addressed by Weiskopf and Ertl [180].

3D Texture Advection – Kao et al. discuss the use of 3D and 4D texture advection for the visualization of 3D fluid flows [69]. The results show sparse texture noise in order to visualize inside the 3D vector field. Formidable challenges are introduced by the memory requirements involved in using 3D and 4D textures. The proposed method does not work well for the case of flows containing critical points for incoming flows from the grid boundary.

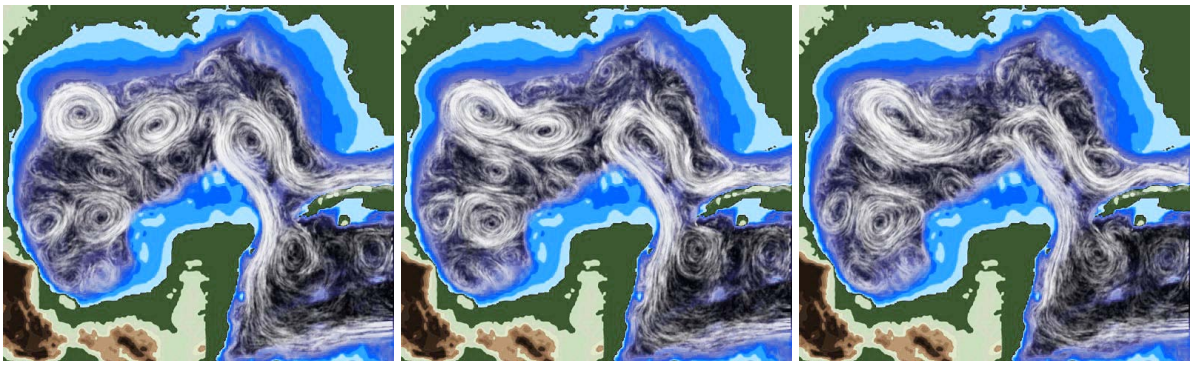


Figure 2.10: Three images taken from an animation of an unsteady vector field created with the Lagrangian-Eulerian advection algorithm. [63, 64] Image courtesy of Jobard et al.

GPU-Based LIC – Heidrich et al. [52] exploit pixel textures to accelerate LIC computation. Pixel textures are an OpenGL extension by SGI that provides the functionality of dependent textures in combination with multi-pass rendering. Heidrich et al.'s implementation supports 2D, steady vector fields only, and achieves sub-second computation times for LIC image generation. While this method could be categorized as a GPU-accelerated LIC technique, we position it here due to its comparability with the following texture advection techniques [62, 181] that use the same proposed OpenGL extension, handle unsteady flow, and thus can be considered an extension of this technique.

LEA – Jobard et al. [62] introduce a GPU-assisted texture advection technique for the dense visualization of 2D, unsteady flow. While the method of Max and Becker [102, 103] advects textures based on coarse triangular meshes, Jobard et al. advect textures on a per-pixel basis by means of pixel textures, which are used in a similar way as by Heidrich et al [52]. The gray-scale texture from the previous time step is dragged along the flow field by modifying the texture coordinates for the dependent texture lookup according to the flow data. Nearest-neighbor sampling is combined with an update of fractional texture coordinates to represent subtexel motion and, at the same time, maintain a high contrast. An iterative injection of additional noise is used to compensate for a possible loss of contrast over time. Jobard et al. also discuss the treatment of inflow at boundaries, image enhancement by color masking, and the use of dye advection. Because of the limited functionality of the graphics hardware that supports pixel textures, the implementation requires many rendering passes and advects a texture of size 256^2 at approximately two frames per second. Moreover, the maximum resolution of textures is restricted to 256^2.

Jobard et al. extend this method to the more flexible Lagrangian-Eulerian Advection (LEA) scheme [63] for the visualization of unsteady, 2D flow. Here, they rely on a CPU implementation that leads to better advection quality, higher speed, and no limitations of the maximum flow size. Particle paths are integrated as a function of time, referred to as the Lagrangian step, while the color distribution of the image pixels is stored in a texture and updated in place (Eulerian step). The temporal coherence of the advected noise textures is transformed into spatial coherence by blending textures from subsequent time steps, i.e., each still frame depicts the instantaneous structure of the flow, whereas an animated sequence of frames still reveals the motion of the advected texture. Jobard et al. demonstrate that the combination of noise and dye advection is useful for an effective visualization and exploration of unsteady flow. Some results from the technique are shown in Figure 2.10. This work is extended by Jobard et al. [64] in order to improve the quality of dye advection.

Weiskopf et al. [179] present a GPU-accelerated version of the LEA algorithm using per-fragment operations. The GPU-based texture advection by Weiskopf et al. [181] supports bilinear dependent texture

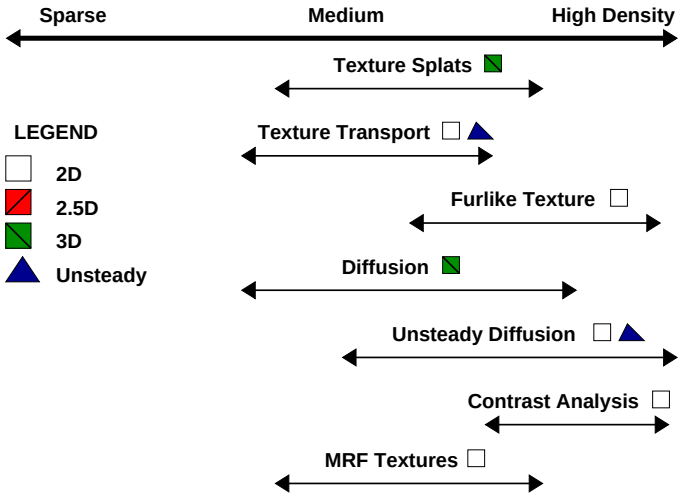


Figure 2.11: Related dense, texture-based flow visualization methods. Each method is compared with respect to the density of the resulting visualization.

lookups without taking into account the update of fractional coordinates. Therefore, this approach is mainly suitable for dye advection at high frame rates. Weiskopf et al. also demonstrate how GPU-accelerated visualization of unsteady, 3D flows can be implemented with pixel textures.

UFAC – Weiskopf et al. [178] introduce a generic texture-based framework for visualizing 2D, time-dependent vector fields. They propose Unsteady Flow Advection-Convolution (UFAC) as an application of the framework for visualizing unsteady fluid flow. Also, their approach can reproduce other techniques such as LEA [64], IBFV [164], UFLIC [140], and DLIC [145]. Weiskopf et al. describe a GPU-accelerated implementation that, among other things, allows the user to trade-off quality for speed.

Related Dense, Texture-Based Methods

The literature described in the following is not, in general, as strongly inter-related as the literature in the spot noise, LIC, texture advection, and GPU-based categories. For this reason we sought an alternative scheme in order to relate the different techniques. Figure 2.11 shows the related methods and classifies them based on the density of their results. In this case each technique is given a subjective rating on a sparse-to-dense scale. Sparse results look more like the results from flow visualization using geometric objects whereas dense techniques produce results resembling spot noise or LIC. These methods do not fit cleanly into one of the previous categories, nonetheless, they are important to the dedicated topic and are briefly outlined here. Reading from top to bottom in Figure 2.11, we visit the techniques in chronological order.

Texture Splats – As an extension of the technique of splatting from volume rendering, Crawfis and Max [25] introduce the notion of *texture splats* for flow visualization. Being a volume rendering technique, it is targeted at the depiction of 3D vector fields. As with Rezk-Salama et al. [121], it is a selective opacity transfer function that ultimately decides which subsets of the 3D data are shown and which are not. The transfer functions are used to emphasize or suppress spatial regions as opposed to ranges of data values.

Texture Transport – The texture transport method of Becker and Rumpf [5] introduces a mathematical framework based on the solution of a time-dependent transport equation. Lagrangian coordinates are

computed from the transport equation and visualized using a texture mapping. The results in this case resemble those from the geometric class of solutions. Individual lines in the texture can be distinguished. The major drawback of this approach is the computation time required.

Furlike Texture – Khouas et al. synthesize LIC-like images in 2D with furlike textures [71]. Their technique is able to locally control attributes of the output texture such as orientation, length, density, and color via a model based on filaments resembling fur.

Diffusion and Unsteady Diffusion – Preußer and Rumpf [119] as well as Diewald et al. [34] borrow a well known technique from image analysis for visualization of fluid flow. The nonlinear, anisotropic diffusion equations from image analysis are adopted and applied to vector fields. A noisy texture covering the domain is strongly smoothed along integral lines while still retaining and enhancing edges in directions orthogonal to the flow, i.e., streamline-aligned diffusion. Successively coarse patterns representing the vector field can also be generated. It is applied to 2D, 2.5D, and 3D vector fields [34, 119]. In the case of 3D, the resulting enhanced edges are discretized and resemble streamlines or streamribbons. In this case, occlusion becomes an important issue because the 3D results appear somewhat cluttered.

Bürkle et al. extend this technique to the case of time-dependent flow [15]. Instead of streamline-like patterns, streakline patterns are generated. A blending strategy, comparable to noise or dye injection, is introduced in order to provide the new time-dependent texture necessary for the case of long-term flow evolution. They propose a solution based on the blending of different results from the transport diffusion evolution started at successively incremented times. Again, the disadvantage of this approach is the required computational time. Also, no attempt is made to apply this method to time-dependent 3D flow, a formidable challenge.

Contrast Analysis – Sanna et al. [131] focus on the issue of encoding another scalar value into the texture used to visualize the flow, in addition to flow direction, orientation, and local magnitude of the field. It is an extension of a previous technique called TOSL–Thick Oriented Streamline Algorithm [130]. Areas of higher scalar values are characterized by higher contrast levels in the texture and streamline tones are generated in order to highlight these areas. The goal is to allow an additional variable into the visualization beyond previous techniques.

MRF – Taponecco and Alexa apply Markov Random Field (MRF) texture synthesis methods to vector fields [148]. The results resemble a mixture of traditional texture-based methods and geometric methods. In the resulting texture, distinct streamline patterns can be seen. One drawback to this method is performance. MRF texture synthesis methods may require hours of computation time. How it may be applied to unsteady flow is an open question.

2.6 Geometric Flow Visualization

Geometric FlowVis entails extracting geometric objects for which their shape is directly related to the underlying data. In what follows, we discuss geometric flow visualization techniques such as contouring in both 2D and 3D as well as geometric FlowVis using integral objects (such as streamlines).

Contouring in 2D – Contouring is a natural extension to color coding in 2D. A contour is a boundary between two distinct regions. Often, the user is highly interested in transition areas in the vector field. In a color plot, transitions are shown by a change of color. With contouring, an explicit line or curve is drawn.

Figure 2.12: Three images from an interactive exploration of a vector field using the MR viewer [68]. A suitable level of resolution can be chosen while maintaining a roughly constant streamline density. Image courtesy of Jobard and Lefer.

Isosurfaces in 3D – Extending contouring from 2D to 3D, results in the use of isosurfaces for 3D flow visualization. Special care needs to be taken with isovalue selection, mostly because of the often smooth nature of flow data. In cases of no sharp transitions within the data, any isovalue lacks (at least partially) intuitive interpretation. Nevertheless there are useful applications of isosurfaces to flow data, e.g., in the visualization of shock waves [147] or burning fronts in simulated combustion data. Furthermore, when scalar clipping is used with color coding of slices, this naturally combines with isosurfaces as long as isovalue and clipping value coincide. Röttger et al. present a hardware accelerated volume rendering technique which allows to use multiple (semitransparent) isosurfaces for visualization [126]. Treinish applies isosurfacing to visualize (unsteady) weather data [157]. Weber et al. [173] present crack-free isosurface extraction for (multiresolution) grids. Laramee and Bergeron provide isosurfaces for super adaptive resolution grids [79]. In this thesis we combine isosurfaces with texture-based flow visualization (Chapter 5 [84]).

Geometric 2D FlowVis Using Integral Objects

In this subsection we shortly discuss geometric FlowVis techniques in 2D based on integral objects such as streamlets, streamlines, and their relatives within unsteady flows. The seeding problem is addressed, which requires a solution in order to realize better distributions of integral objects.

Streamlets and Streamlines in 2D – If flow vectors are integrated for a very short time, streamlets are generated. Even though short, streamlets already communicate temporal evolution along the flow. If longer integration is performed (as compared to streamlets), streamlines are gained. They are a natural extension of glyph-based techniques and offer intuitive semantics: users easily understand that flows evolve along integral objects.

Streaklines, timelines, and pathlines – When unsteady flow data are investigated, several distinct integral objects are used for flow visualization. A pathline or particle trace is the trajectory that a particle follows in a fluid flow [136]. A timeline joins the positions of particles released at the same instant in time from different insertion points, i.e., joins points at a constant time t [109]. A streakline is traced by a set of particles that have previously passed through a unique point in the domain [136]. Streaklines relate to continuous injection of foreign material into real flow. Sanna et al. present an adaptive visualization method using streaklines where the seeding of streaklines is a function of local vorticity [128].

Streamline seeding in 2D – One important aspect of streamlines, or integral curves, when used for visualizing continuous vector fields is the best choice of initial conditions. Since, in general, evenly distributed seed points do not result in evenly spaced streamlines, special algorithms need to be employed. Turk and Banks [159] as well as Jobard and Lefer [65] developed techniques for automatically placing seed points to achieve a uniform distribution of streamlines on a 2D vector field. Streamline seeding strategies in 2D may also be topology based. Verma et al. [169] present a seed placement strategy for streamlines based on flow features in the data set. Their goal is to capture flow patterns near critical points in the flow field. Building on their previous work, Jobard and Lefer presented a multiresolution (MR) method for visualizing large, 2D, steady-state vector fields [68]. The MR hierarchy supports enrichment and zooming. The user is able to interactively set the density of streamlines while zooming in and out of the vector field (Figure 7). The density of streamlines can be computed automatically as a function of velocity or vorticity.

Seeding of integral objects becomes a special challenge when dealing with time-dependent data. Jobard and Lefer presented an unsteady FlowVis algorithm by correlating instantaneous visualizations of the vector field at the streamline level [67]. For each frame, a feed forward algorithm computes a set of evenly-spaced streamlines as a function of the streamlines generated for the previous frame. Their method also provides full control of the image density so that smooth animations of arbitrary density can be produced.

FlowVis Using Geometric Objects on Slices or Boundaries

After discussing 2D FlowVis based on geometric objects, this subsection shortly addresses similar approaches on subsets of 3D flows such as boundary flows. Interpretation of integral curves on sectional slices requires special care.

Integrated tufts – Wegenkittl et al. use integrated tufts (similar to streamlets), seeded on specific equilibrium surfaces, for the visualization of a complex dynamical system [174], also over variations of that system in a fourth dimension.

Geometric objects on slices or boundaries – Similar to 2D FlowVis, geometric objects such as streamlines are also used for visualizing boundary flows or sectional slices through 3D flow [42]. However, it is important to note that the use of these objects on slices may be misleading, even within steady flow data sets. A streamline on a slice may depict a closed loop, even though no particle would ever traverse the loop. The reason again lies in the fact, that flow components which are orthogonal to the slice are omitted during flow integration.

Streamline seeding on boundary surfaces – Mao et al. [99] extend the streamline seeding of Turk and Banks [159] in order to generate evenly distributed streamlines on boundary surfaces within curvilinear grids.

3D FlowVis Using Geometric Objects –

When dealing with 3D flow, a rich variety of geometric objects is available for flow visualization. This subsection addresses a series of objects, from streamlets to flow volumes, primarily sorted according to their dimensionality, and within equal dimensionality roughly with respect to which technique extends to another.

Streamlets in 3D – Streamlets easily extend to 3D, although perceptual problems may arise due to distortions resulting from the rendering projection. Also, seeding becomes more important in 3D, again. Löffelmann and Gröller use a thread of streamlets along characteristic structures of 3D flow to gain selective, but importance-based seeding as well as an enhancement of abstract flow topology through direct visualization cues [91].

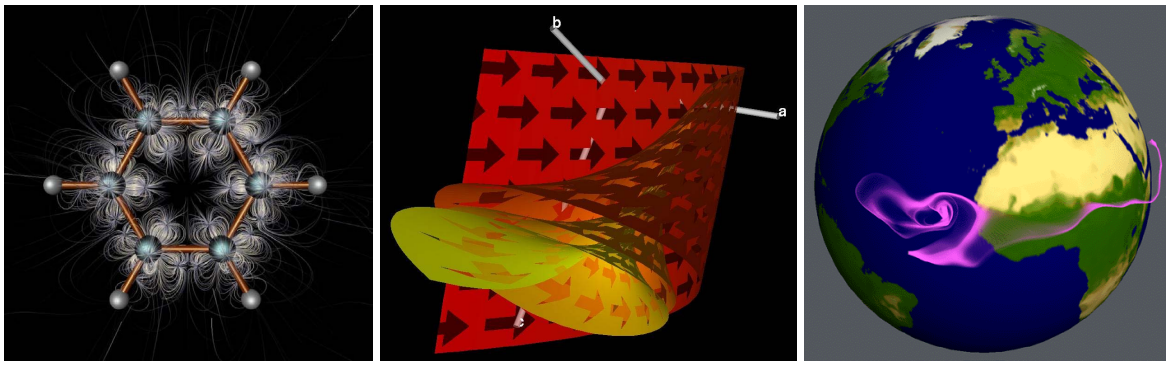


Figure 2.13: Examples of flow visualization using geometric objects (left) illuminated streamlines [186], (middle) stream arrows [95], and (right) flow volumes [104].

Streamlines in 3D – At NASA the Flow Analysis Software Toolkit (FAST) [2] is used to visualize CFD data based on streamlines in 3D. Careful seeding is necessary to obtain useful results, since visual clutter can easily become a problem. Illuminated streamlines Zöckler et al. present illuminated streamlines to improve perception of streamlines in 3D by taking advantage of the texture mapping capabilities supported by graphics hardware [186]. Their shading technique increases depth information. By making the streamlines partially transparent, they also address the problem of occlusion, as shown in Figure 2.13, left). For seeding, the authors propose an interactive seeding probe which can be moved around to start streamlines at specific places of interest. Also, seeding near potential objects of interests is demonstrated.

Particle tracing in 3D – Kenwright and Lane present an efficient, 3D particle tracing algorithm that is also accurate for interactive investigation of large, unsteady, aeronautical simulations [70]. A performance gain is obtained by applying tetrahedral decomposition to speed up point location and velocity interpolation in curvilinear grids.

Teitzel et al. analyse different integration methods in order to evaluate the trade-off between time and accuracy [150, 152]. They present a 3D particle tracing algorithm targeted at sparse grids that is very efficient with respect to storage space and computing time. The authors recommend using sparse grids as a data compression method in order to visualize huge data sets.

Nielson presents efficient and accurate methods for computing tangent curves for 3D flows [108]. The methods work directly with physical coordinates, eliminating the need to switch back and forth to computational coordinates. Efficient particle tracing methodologies are also addressed by Sarajane et al. [127]. Since streamlines are usually easily computed in real time, they offer (together with their intuitive semantics) an often chosen tool for interactive flow analysis. Bryson and Levit [14] demonstrate seeding of integral objects in a virtual 3D environment by use of a rake.

Stream ribbons and streamtubes – A first extension of streamlines in 3D are stream ribbons and streamtubes. A stream ribbon is basically a streamline with a winglike strip added, to also visualize rotational behavior of the 3D flow (which is not possible with streamlines alone) [160]. A streamtube is a thick streamline that can be extended to show the expansion of the flow [160]. Stream ribbons and streamtubes offer advantages over streamlines in that they can encode more properties, such as divergence and convergence of the vector field, in the geometric properties of the respective integral objects.

Ueng et al. present techniques for efficient streamline, stream ribbon, and streamtube constructions

on unstructured grids [160]. A specialized Runge-Kutta method is employed to speed up streamline computation. Explicit solutions are calculated for the angular rotation rates of stream ribbons and the radii of streamtubes. The resulting speedup in overall performance aids in the exploration of large flow fields.

Fuhrmann and Gröller [47] use dash tubes, i.e., animated, opacity-mapped streamtubes, as a visualization icon. An algorithm is described which places the dash tubes evenly in 3D space. They also apply a magic lens and magic box as interaction techniques for investigating densely filled areas without filling the image with visual detail and complexity.

Laramee introduces the streamrunner as an extension of streamtubes an interactively controlled 3D flow visualization technique that attempts to minimize occlusion, minimize visual complexity, maximize directional cues, and maximize depth cues by letting the user control the length of the streamtubes [77].

Stream polygons – Another extension of streamlines are stream polygons used by Schroeder et al. [134]. Stream polygons are tools to visualize vectors and tensors using tubes with a polygonal cross section. The properties of the polygons such as the radius, the number of sides, the shape and the rotation reflect properties of the vector field including strain, displacement, and rotation.

Streamballs and streakballs – Streamballs are a useful flow visualization technique used by Brill et al. [13], which visualizes divergence and acceleration in fluid flow. Streamballs split or merge depending on convergence/ divergence and acceleration/deceleration, respectively. Teitzel and Ertl introduce streakballs when they present and compare two different approaches to accelerate particle tracing on sparse grids and curvilinear sparse grids for unsteady flow data [149].

Stream surfaces – Yet another extension to streamlines are stream surfaces, which are surfaces that are everywhere tangent to a vector field. A stream surface can be approximated by connecting a set of streamlines along timelines (and varying the number of streamlines used according to convergence or divergence of the flow) [59]. Stream surfaces are very good for texture based visualization techniques such as Spot Noise and LIC, because there is no cross-flow component normal to the surfaces, i.e. the vector field is not projected like it is for 2D slices through a 3D domain [91]. Stream surfaces present challenges related to occlusion, visual complexity, and interpretation.

Hultquist presents an interactive flow visualization technique using stream surfaces [58]. Van Wijk presents two follow-up techniques for generating implicit stream surfaces [167]. Cai and Heng [18] address the issues associated with the placement and orientation of stream surfaces in 3D.

Löffelmann et al. present stream arrows (see Figure 2.13, middle) as an enhancement of stream surfaces by separating arrow-shaped portions from a stream surface [94, 95]. Stream arrows address the problem of occlusion associated with 3D flow visualization, but especially with stream surfaces. Stream arrows also provide additional information about the flow, usually not seen with stream surfaces, such as flow direction, convergence/divergence, et cetera.

Van Wijk simulates stream surfaces by a large set of so-called surface particles [166]. Surface particles exhibit less occlusion when compared to stream surfaces. Interestingly, Van Wijk s approach in a way anticipated recent advances in pixel-based rendering techniques.

Time surfaces in 3D – A natural extension of timelines (in 2D or 3D) are time surfaces, when constant-time instants of moving particles are assumed, which previously have been released from a two-dimensional patch. An example of an application of this principle, are level-set surfaces used by Westermann et al. [183].

Flow volumes – The last (direct) extension of a streamline into 3D described here are flow volumes (see Figure 2.13, right). A flow volume is a specific subset of a 3D flow domain, which is traced out by a particular initial 2D patch over time as described by Max et al. [104]. The resulting volume is divided up into a set of semitransparent tetrahedra, which are volume rendered in hardware in a way derived from the method of Shirley and Tuchmann [141]. Becker et al. extend flow volumes to unsteady flow [4]. The resulting unsteady flow volumes are the 3D analogue of streaklines. Considerations are made when extending the visualization technique to unsteady flows since particle paths may become convoluted in time. The authors present some solutions to the problems which occur in subdivision, rendering, and system design. The resulting algorithms are applied to a variety of flow types including curvilinear grids.

2.7 Comparisons and Discussion

In this section we briefly introduce literature that compares and discusses dense, texture-based techniques at a meta-level. Sanna et al. also provide a summary of this area of research, with a different classification [129]. The methods are classified according to the dimensionality outlined here in Section 2.1.

Flow Textures – Erlebacher et al. [41] present a class of flow visualization algorithms called *flow textures* within a common conceptual framework. Flow textures are textures that encode dense, 2D, time-dependent representations of flow. The framework allows important ingredients of flow texture algorithms to be understood with respect to spatial and temporal correlation. A subset of the more recent visualization techniques is described.

User Studies – Laidlaw et al. [75] present one of the few findings related to human-computer interaction (HCI). They attempt to assess some different visualization techniques from the viewpoint of the user in terms of searching for and classifying critical points in the flow and predicting where a particle may end after advection. Error was highest for the LIC technique in conjunction with classifying critical points and the prediction of particle advection. This is probably due to the fact that LIC images do not distinguish between upstream and downstream flow. User error was higher than expected for all methods. Hedgehog techniques and LIC were also associated with high error for locating critical points. The authors postulate that this was because in many cases critical points near the borders of the vector field were difficult to identify.

2.8 Discussion and Future Prospects

Texture-based and geometric flow visualization algorithms are effective, versatile, and applicable to a wide spectrum of applications. A large number of techniques have been developed and refined. In general, which techniques are best depends strongly on the goal of the visualization, such as for exploration, detailed analysis, or presentation and on the kind of data involved. Therefore, we believe that a large variety of techniques should be available in order to allow researchers to choose the most suitable one.

The problem of dense, 2D, unsteady flow visualization is close to being solved [164]. And with recent follow-up work [83, 85, 165], unsteady flow visualization on surfaces is not far behind. However, the generalization to 3D flow fields is still unsolved, especially in the case of unsteady flow. Hardware, arguably, will not be the primary bottleneck to solving this challenge, but perceptual issues will. Perceiving three spatial and three data dimensions directly is a difficult job for the visual and cognitive system. So far, techniques based on geometric objects and particle animation generalize better to 3D fields.

The scale of numerical flow simulations, and thus the size of the resulting datasets, continues to grow rapidly–generally faster than the size of computer memory. For these reasons more simplification strate-

gies must be conceived, such as spatial selection (slicing, regions of interest), geometry simplification, and feature extraction.

Slicing in a 3D field reduces the problem to 2D, allowing the use of good 2D techniques, but care must be taken with interpretation, as the loss of the third dimension may lead to physically irrelevant results and wrong interpretation. Taking a single 3D time slice from a 3D time-dependent dataset has similar dangers. Other spatial selections such as 3D region-of-interest selection are less risky, but may lead to loss of context. Reduction of data dimension, such as reducing vector quantities to scalars will give more freedom of choice in visualization techniques (such as volume rendering), but will not lead to much data reduction. Geometry simplification techniques such as polygon mesh decimation, levels-of-detail, or multiresolution techniques will be effective in managing very large datasets and interactive exploration, enabling users to trade accuracy with response time. Some areas that need additional work are:

- dense visualization techniques in 3D
- multi-field visualization with scalar, vector, and tensor data,
- handling and exploring huge time-dependent flow datasets,
- user studies for evaluation, validation, and field testing of flow visualization techniques,
- visualization of inaccuracy and uncertainty [90, 124],
- more robust feature extraction techniques, especially in the case of 3D flow.

We also note that much of the research literature presented here demonstrates methods operating on structured, uniform resolution grids. However, the grids used in the private, commercial industry sector are often adaptive resolution and unstructured, especially in the case of CFD [78, 83]. Thus further research is necessary in order to integrate many of the these methods into practical industrial applications. For supplementary material, including additional, higher resolution images and animations, please visit: `http://www.VRVis.at/ar3/pr2/star/`

Chapter 3

FIRST: A Flexible and Interactive Resampling Tool for CFD Simulation Data

"I never think of the future - it comes soon enough."

– Albert Einstein [1] (1879–1955)

The demand for analysis and 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 software simulation tools to analyze design decisions rather than constructing real, heavyweight objects. At the VRVis Research Center we collaborate with AVL in order to provide visualization solutions for analysis of their CFD simulation result data. AVL (`www.avl.com`) is an internationally recognized leader in providing automotive design and CFD simulation solutions to its partners in the automotive industry. AVL works with other internationally recognized companies such as Toyota, DaimlerChrysler – Mercedes-Benz, and Pierburg Instruments GmbH.

3.1 Flexible Tools for Unstructured Grids

Figure 3.1 shows two intake ports -small valves in a car engine that allow air into the engine's cylinders. The source of the flow is the cubical structure on the left. Flow travels through the connecting pipes in the middle, down through the intake ports, and is ignited in the combustion cylinder. Figure 3.2, a close up image of the intake ports, helps reveal the adaptive levels of resolution contained in the mesh. These data sets require flexible analytic and visualization solutions. Ideally, the tools used to analyze and visualize these data sets should be flexible enough to adapt their *size*, *shape*, *orientation*, and *resolution* to fit the individual components of the data sets either automatically or through user-specified parameters. The resampling tool we present, called FIRST (a *F*lexible and *I*nteractive *R*e*S*ampling *T*ool), meets precisely these requirements.

A Wide Variety of Simulation Data Sets

The tools used to analyze and visualize these data sets have to address not only the versatility of individual geometries but also the *wide range* of simulation data sets that undergo analysis. AVL works with a large, varied collection of data sets ranging from small geometries such as small fluid conduits and

[1] German physicist, discovered special relativity 1905 and general relativity 1915-1916, explained photoelectric effect and Brownian motion, Nobel Prize in Physics 1921

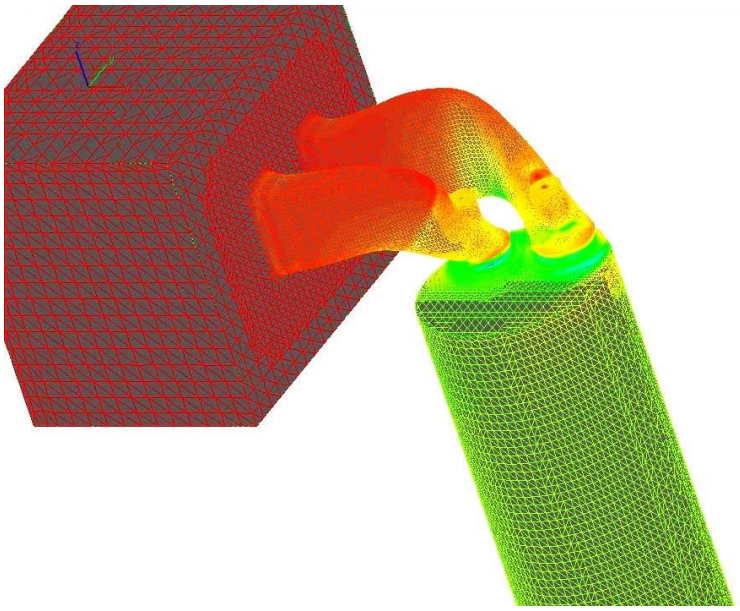


Figure 3.1: The CFD simulation grid of an intake port. The image illustrates the versatility of a typical, unstructured, CFD simulation grid containing a flow source on the left, two connecting pipes in the middle, two intake ports at the ends of the pipes, and a combustion chamber on the right.

cylinders to mid-range size geometries such as cooling jackets, intake manifolds, catalytic converters, to large geometries such as automotive cabin interiors and automotive exteriors. The geometric sizes of these grids, as well as the sizes of the underlying polygons, differ by *six* or more orders of magnitude. Furthermore, we speculate that this difference will only increase in the future.

The unpredictable nature of both the data sets and the individual components the engineer may choose to analyze provide strong motivation for tools that offer many options and, thus, higher levels of control to the user. The tools used to visualize the simulation results also need to span this range of sizes correspondingly. The FIRST tool described here offers six interactive *degrees of freedom* (DoF) with the goal of addressing the nature of both the wide range of data sets and the interests of engineers.

Perceptual Problems in FlowViz

When analyzing the results of a CFD simulation, engineers are interested in the option of examining planar vector fields with normal components [132]. An obvious approach to this problem is to place glyphs (e.g. directed lines, arrows, cones etc.) oriented in the direction of the vector components, at selected points of the vector field. Ideally, the length of the glyphs should be equal to the normal of the vector, but often a scaling constant is added to improve the visualization result. This method (called hedgehog visualization) [136] is illustrated in Figure 3.3.

There are multiple drawbacks to this approach. *Perceptual problems* such as visual complexity and occlusion can result. Figure 3.3 shows a 2D slice taken through the intake port data set with the surface also shown as semi-transparent context information. Here, the glyphs are either too big, resulting in occlusion, or too small to clearly indicate directional information. Another problem arises due to the number and *placement* the glyphs. Critical points might be missed if they are either occluded by too many glyphs placed in the wrong areas or if an insufficient number of glyphs is chosen in certain areas. FIRST solves both the perceptual and placement problems by (1) giving the user control of the *resolution* of the glyphs in the image and (2) giving the user precise control of *where* to place the vector glyphs for

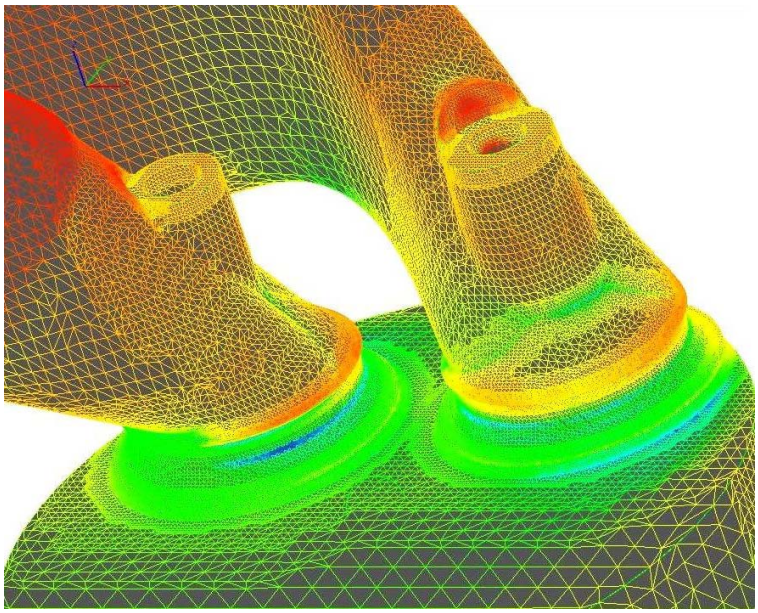


Figure 3.2: A close-up view of the surface of the intake port(s) shown in Figure 3.1. We can see many components, features, and multiple, adaptive resolution levels of unstructured grid cells. This versatility motivates interactive tools that have the ability to change their size, shape, orientation, and resolution to match the component(s) of the mesh currently undergoing user analysis.

viewing the flow with normal components.

The rest of the chapter is organized as follows: Section 3.2 describes related research work. Section 3.3 describes the contribution of this work. The resampling algorithm and user options are described in Section 3.4. Timing and results are presented in Section 3.5. Conclusions are drawn in Section 3.6.

3.2 Related Work in Resampling

Since we did not cover literature related to resampling in Chapter 2, we briefly outline some related literature here. When describing grids, we follow the terminology of Yagel et al. [185]. *Structured grids*, including *cartesian grids*, *regular grids*, and *rectilinear grids*, all maintain an implicit neighborhood connectivity (Figure 2.1(a-d)), i.e. the position of grid cells can be computed, rather than stored explicitly. In contrast, grids whose neighbors must be explicitly stored are categorized as *unstructured* grids. This distinction is important in understanding our claim that the resampling technique here can be used to represent *any* unstructured grid with *any* structured grid. We demonstrate this assertion with implementations of the resampler from unstructured triangular grids (Figure 2.1(f)) to cartesian and polar grids (Figure 2.1(a-d)). The basis for this claim hinges on the supposition that cell positions in structured grids can be computed.

The literature dedicated to the topic of resampling spans multiple, sometimes disparate, goals. While we designed our resampler with the goal of a high level of user-control, other resampling techniques focus on the goals of volume rendering, surface reconstruction, and accurate surface normal representation. In contrast with these approaches, we concentrate on 2D slices through a 3D mesh from CFD. We contrast other resampling goals with ours.

Resampling techniques whose goal is to achieve faster volume rendering speeds may make use of the hardware capabilities offered by modern graphics cards. Westermann presents a resampling technique

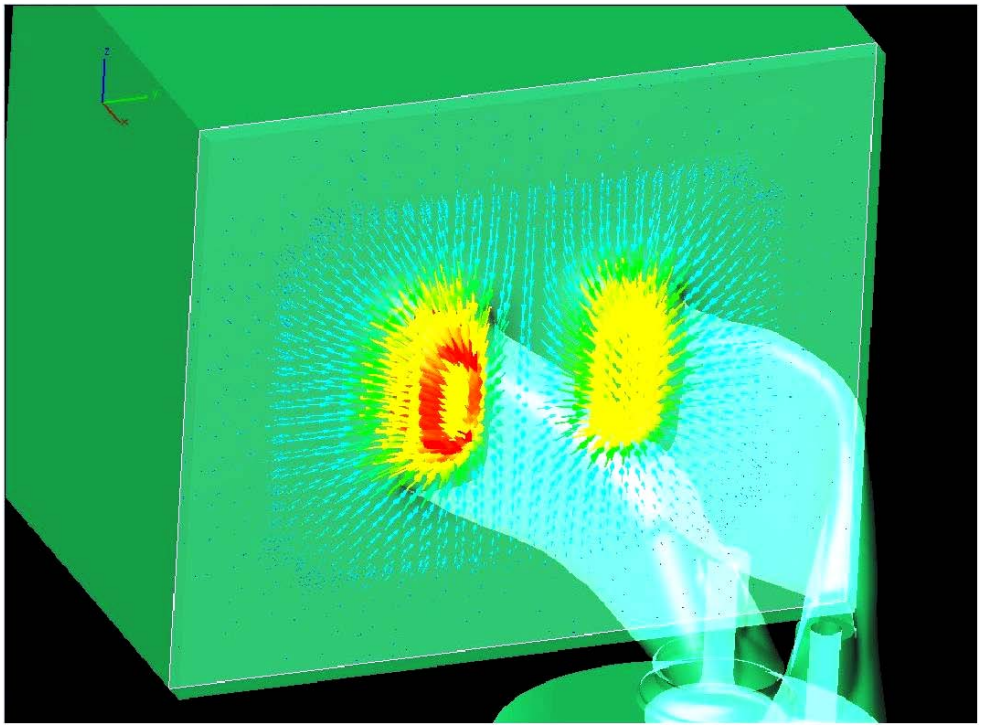


Figure 3.3: Visualizing the direction of the flow, including the normal component, using the classic hedgehog visualization technique can result in perceptual problems such as visual complexity and occlusion. Here the problems are illustrated in a slice through the intake port data set with semi-transparent context information.

for resampling scalar fields given on unstructured tetrahedral grids [182]. The goal of this research is to bring direct volume rendering towards interactive frame rates with the assistance of graphics hardware. Determining the visibility ordering of grid elements is a major challenge in this scenario. While the goal of this approach is dissimilar to ours, one similarity worthy of note is that Westermann's technique can be used to display time-dependent unstructured grids with changing geometry and topology.

Weiler and Ertl [177] also present three resampling approaches and contrast the results with the work of Westermann [182]. Their goal is to minimize performance time via balancing the computing workload between the native processor and the graphics board processor.

In addition to resampling techniques for volume rendering, some resampling approaches are targeted at representing surfaces. Botch and Kobbelt [12] present a technique to resample feature regions of a given triangle mesh with the goal of reducing aliasing artifacts on surfaces, i.e., surface anti-aliasing. Similar the work here, there is a strong interaction component to their work. The user chooses the areas on the surface to which the surface normal anti-aliasing algorithm is applied. To re-model the structure of a sample CFD grid can take an interactive session lasting one hour.

Rocchini et al. [125] present an algorithm whose goal is the removal of small topological inconsistencies and high frequency details from surfaces. One of their goals is the simplification of huge meshes, i.e., meshes composed of millions of faces.

It is also worthy of note that some resampling algorithms from the image processing domain magnify or minify the original image via sampling at regular intervals [120]. Our approach samples the original data irregular intervals and generates an evenly-spaced representation. We also note that the material in this chapter has also been published elsewhere [78].

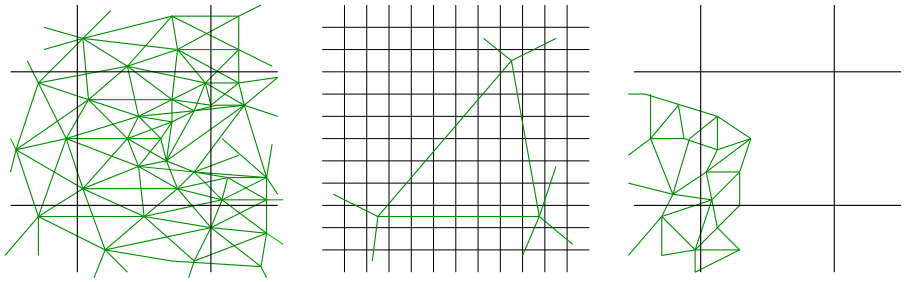


Figure 3.4: The FIRST algorithm handles 3 cases: (I) when several scalene triangles influence one resampler cell, (II) when one triangle influences several resampler cells, and (III) when several triangles influence one resampler cell, however, the resampler cell is not rendered. The resampler cells are outlined in black while the original unstructured grid is drawn colored.

3.3 Interactive Visualization and Analysis

The key distinguishing features of FIRST stem from the fact that it was specifically developed in order to provide the user with a range of flexible interactions at multiple resolutions. The reason we focus on a combination of user control with resampling is because the engineers at AVL require interactive visualization solutions. We speculate that this is due to a large amount of time engineers spend when investigating the data sets. 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 measured, experimental results. FIRST has the following advantages:

1. Six interactive DoFs: three translational, scaling, rotation, and resolution
2. handles changes to both underlying topology and geometry, i.e., can be utilized for the display of time-dependent, unstructured grid slices where geometry and topology change over time or space (Section 3.4)
3. resamples any unstructured grid (Figure 2.1(f)) onto any structured grid (Figure 2.1(a-d))
4. handles unstructured grids with holes and discontinuities (Figure 3.10)
5. does not rely on any pre-processing of the data
6. consists of a straightforward implementation, e.g., requires no neighbor-finding capabilities or complicated data structures
7. processes large quantities of unstructured, scalene triangles efficiently

The resampler we describe provides flexible user-interaction capabilities beyond those offered by previous methods. Also, the algorithm operates on a per-unstructured-polygon basis, making it suitable for parallelization.

3.4 FIRST: A Flexible and Interactive ReSampling Tool

Because we are dealing with unstructured, adaptive resolution grids and because the users require the option of fine resampling, the resampling algorithm presented here is required to handle a minimum of three cases (Figure 3.4): (I) the underlying unstructured grid cells are generally smaller than the structured resampler cells, (II) the unstructured grid cells are generally larger than the resampler cells, and (III) some unstructured grid cells are inside a resampler cell, however, the resampler cell should not

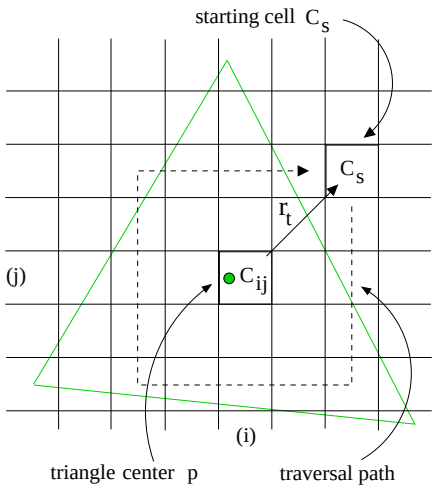


Figure 3.5: A schematic of the resampling algorithm. A loop of structured grid cells is tested for influence by the unstructured grid cell.

be rendered. A resampler cell is rendered only if its center is covered by the underlying unstructured grid. In this paper we use the term *resampler cell* to refer to a structured grid cell that represents, or summarizes, the original data.

Algorithm and Implementation

It is important to note that, unlike many other applications where vector field values are stored at triangle vertices, the vector field here is associated with the centers of the triangles in the slice mesh. By slice mesh, we mean the user-defined 2D slice through the original 3D mesh. Engineers require the option of visualizing the *original* values resulting from the simulation rather than interpolated values at the triangle vertices. Storing the CFD simulation data values at the center of the grid cells may result from using a finite volume method to solve for the flow quantities [76].

We have, however, implemented the option of viewing the interpolated scalar values at the triangle vertices. We note that the algorithm here is easily modified to handle unstructured grids where the scalar field is stored at the vertices.

Algorithm Overview

To describe the algorithm, we introduce the notion of *influence*, similar to the notion introduced by Rhodes et al. [122, 123]. A polygon *influences* a resampler cell if the polygon contributes to the summary (or resampled) data of a structured grid cell. This happens when a triangle is in the vicinity of a resampler cell. In short, the resampling algorithm traverses multiple loops of resampler cells in the neighborhood of each unstructured cell in order to see if they are influenced by the original unstructured grid cells. In the most common case (Figure 3.4, Case I), we can think of a resampler cell as summarizing multiple underlying unstructured grid cells.

Algorithm Detail

The following high level pseudo-code summarizes the resampling algorithm (Figure 3.5):

resample():

```
FOR each scalene triangle, T
  compute center point, p, of T
```

```
  compute resampler cell index, C_ij(p)
  int r_t = 0
  bool influence = FALSE
  do
   influence = computeInfluence(T, r_t)
   r_t++
  while (influence == TRUE)
```

The resampling algorithm requires exactly *one* visit to each unstructured grid cell. In the **resample()** method, first the resampler cell, $C_{i,j}$ in Figure 3.5, surrounding the triangle center is computed. Next, a loop of resampler cells is traversed clockwise, starting with cell, C_s. The size of the loop traversed is determined by a topological radius, r_t, starting with a loop radius of 0, and incremented until the triangle, T, is found to have no influence on the loop of resampler cells.

computeInfluence(triangle T, index r_t):

```
  compute resampler cell index, C_s(r_t)
  bool triangleInfluence = FALSE
  FOR each resampler cell in loop
   IF (testInfluenceOfTriangle(T, C_s))
     triangleInfluence = TRUE
return triangleInfluence
```

In the **computeInfluence()** method, each resampler cell in the loop (shown as a dashed line in Figure 3.5) is tested for influence by the current triangle, T, in the unstructured grid. A sample traversal loop is shown in Figure 3.5. A triangle influences a loop when any resampler cell in the loop tests positive for influence.

testInfluenceOfTriangle(triangle T, cell C_s):

```
  bool influence = FALSE
  IF center(C_s) bounded by T
    influence = TRUE // Cases I & III
    render C_s = TRUE
    add V_T to C_s
  IF center(T) bounded by C_s
    influence = TRUE // Case II
    add V_T to C_s
return influence
```

In the **testInflunceOfTriangle()** method, two simple tests are performed: (1) if the resampler cell center is bounded by the triangle cell and, (2) if the triangle center is bounded by the resampler cell. If the resampler cell center is bounded by the triangle, the velocity vector of the unstructured, scalene triangle, V_T is added to the resampler cell's summary vector and the resampler cell is rendered (during the rendering pass). By rendered, we mean simply that a vector glyph is drawn at the center of the resampler cell. If the triangle center is bounded by the resampler cell, the velocity vector of the unstructured, scalene triangle, V_T is added to the resampler cell's summary vector; however, the resampler cell is not necessarily rendered. These two tests cover all three cases shown in Figure 3.4. It is helpful to note that a triangle cell, T, may influence a resampler cell, C_S, without C_S being rendered.

In the actual implementation, the test to see if a resampler cell center falls within the bounds of an unstructured, scalene triangle includes an explicit test of whether the resampler cell center falls precisely on the edge of a triangle. This is because often, the triangles in the mesh are derived from highly

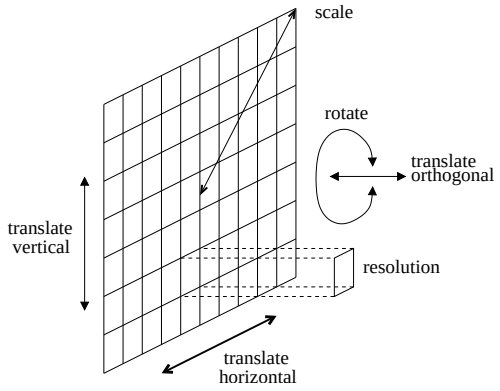


Figure 3.6: FIRST is a six DoF tool: (1-3) three translations, (4) a rotation, (5) a scale, and (6) resolution.

structured portions of CFD components–such as the flow source shown in the left of Figure 3.1.

The resampling algorithm generalizes to other types of unstructured grids, not just those composed of scalene triangles. The reason is two-fold. First, the algorithm operates on a per-unstructured-grid-cell basis making any search for unstructured grid cells unnecessary. Second, given any location in space, such as the center point of an unstructured grid cell, it is easy to compute a shell of regular grid cells around that location. The position of regular grid cells can be computed by definition.

Some nice consequences of the algorithm are that (1) *no* special boundary conditions are checked during the computation and (2) *no* knowledge of the underlying grid's resolution is required. In other words, FIRST does not have to make any distinction between the three cases shown in Figure 3.4. This lays the groundwork for the claim that the algorithm is characterized by ease of implementation.

Interactive Resampling Options

FIRST provides the user with 6 interactive degrees of freedom (Figure 3.6): (1-3) translation along three dimensions, (4) rotation about the center, (5) scaling, and (6) the resolution of resampling cells, i.e., cells/m^2.

The resampler features are associated with a user-defined, 2D slice through a 3D mesh from CFD. Engineers take a slice of the data and slide the slice through the geometry in order to find features of the simulation data, e.g., areas of extreme pressure and vortices. As the user moves the slice through the 3D mesh, the resampler automatically resizes itself around the slice boundary, handling changes to both the underlying geometry and topology. This is important with respect to addressing the versatility aspect of our CFD simulation data sets. Furthermore, requiring the user to manually adjust the size of the resampling grid would slow down the visualization and analysis process considerably.

In addition to the ability to define slices parallel to the X-Y, X-Z, and Y-Z coordinate planes, the user may also define arbitrary cutting planes in 3D space. The user may click on any three locations on the CFD grid and the three points are used to define an interactive slice. For this capability, the unstructured grid cells are sorted into an octree so coordinate searching is fast. All resampling options are available for arbitrary slices.

Discrete Polar Resampling

The FIRST algorithm detailed in Section 3.4 easily generalizes to any structured grid. Figure 3.5 on page 39 illustrates the algorithm applied to a Cartesian grid while Figure 3.7 on page 42 illustrates the same algorithm applied to a polar grid. The only difference is the use of polar coordinates instead of

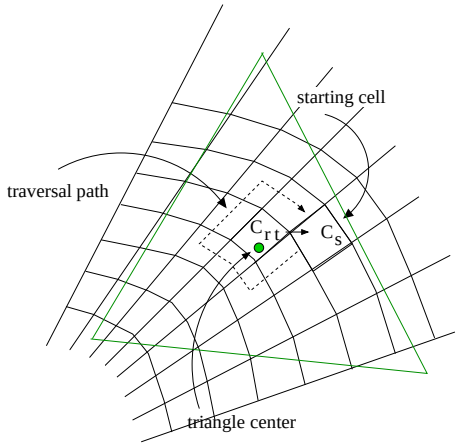


Figure 3.7: The resampling algorithm applied to a polar resampling grid. Polar resampling is a natural application for many of the cylindrical components common in CFD simulation models.

cartesian coordinates. The traversal path is a function of both radius and degrees. And, the resolution of the polar grid is specified in terms of slices and rings.

We have implemented both Cartesian and polar versions of the algorithm, both with identical user options. The reason for offering polar resampling lies in the cylindrical structure of many of the components from common CFD simulation models. The combustion chamber shown in Figure 3.1 on page 35 is a common example of such a component.

Interactive Visualization Options

Multiple visualization options are associated with FIRST including: (1) the ability to toggle wire-frame or semi-transparent context information, i.e., the 3D grid from which slices are defined and (2) several interactive glyph options. This includes any combination of glyph shapes, e.g., cones vs. arrows with rendering options, e.g., wire-frame vs. solid. The user also defines the glyph scaling options that may be used to avoid the "visualization lie" [158]. Several other FlowViz options are provided directly on unstructured grids [77, 83].

Speed vs. Accuracy

If we take a closer look at the resampling algorithm, we note that we make a trade-off between speed and accuracy. A straightforward averaging scheme is used to summarize the vectors influencing a resampler cell. When the resolution of the resampling grid approaches or exceeds that of the underlying, unstructured grid, our algorithm amounts to nearest neighbor interpolation scheme, or a box filter when described in the frequency domain [8, 9]. Reconstructing the flow field at a higher resolution than the original simulation results is orthogonal to the goal of this tool. Users make this trade-off for interactivity because of their tasks related to searching the flow field. Engineers spend time visually searching for areas of extreme pressure, symmetries in the vector field, and critical points. We speculate that users require searching tasks to be as fast as possible. Only after the features of interest are found is more detail required. We offer three options to users requiring higher accuracy: (1) The user may interactively increase the resolution of the resampling grid, thus increasing its accuracy. (2) Users may view the unstructured grid directly, displaying only the original values resulting from the simulation. (3) We have also implemented a *dynamically annotated user dialog* not dissimilar to that of Loughlin and Hughes [98]. This feature allows the user to click anywhere on the slice of interest. Then a dialog

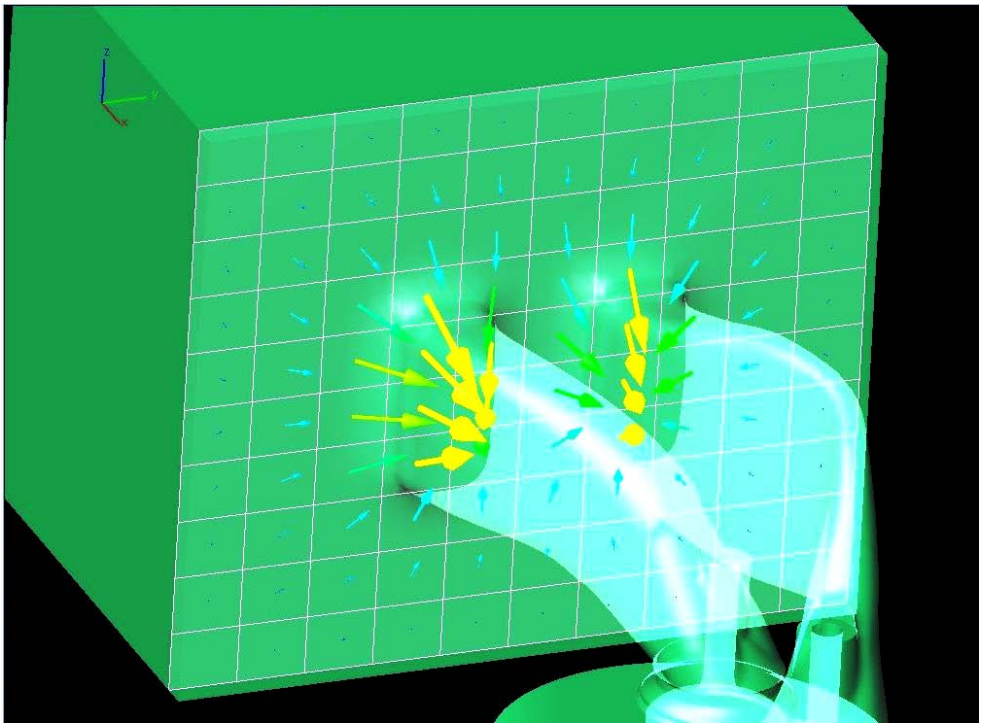


Figure 3.8: Visualizing the direction of the flow, including the normal component, using FIRST. (Compare with Figure 3.3) The resampling grid is outlined in white.

will automatically display the original simulation result values of each variate (e.g. velocity, pressure, temperature, etc.).

3.5 Results

Figure 3.8 on page 43, Figure 3.9 on page 44, and Figure 3.10 on page 46 illustrate an example of viewing a vector field with normal component using the resampler. [2] We can compare Figure 3.8 on page 43 with Figure 3.3 on page 37 and see that viewing the normal components of the vector field is easier using summary vectors rather than the brute force hedgehog technique. Perceptual problems such as occlusion and visual complexity have been greatly reduced. Furthermore, visualization algorithms operating on regular grids are generally faster than on unstructured grids because particle tracing algorithms no longer require neighbor-finding techniques and the more costly flow reconstruction methodology.

Figure 3.10 on page 46 illustrates our technique on a mesh with discontinuities. The discontinuities are two gaps in the shape of rings. Visualization of flow with normal components is shown using both the hedgehog technique versus the glyphs onto a resampled grid. A close-up view of one of the discontinuities is provided.

In order to achieve our goal of implementing a flexible and interactive resampler, interactive frame rates must be achievable. We have tested our algorithm on typical slices of CFD simulation data. Some sample performance times (measured in frames per second) are shown in Table 3.1 on page 45. The performance times depend on both the number of scalene triangles composing the slice and the number of resampler cells. We can see from the table that this resampling algorithm does support interactive frame rates

[2] Supplementary animations of the resampler can be found at
`http://www.VRVis.at/ar3/pr2/resampler/`

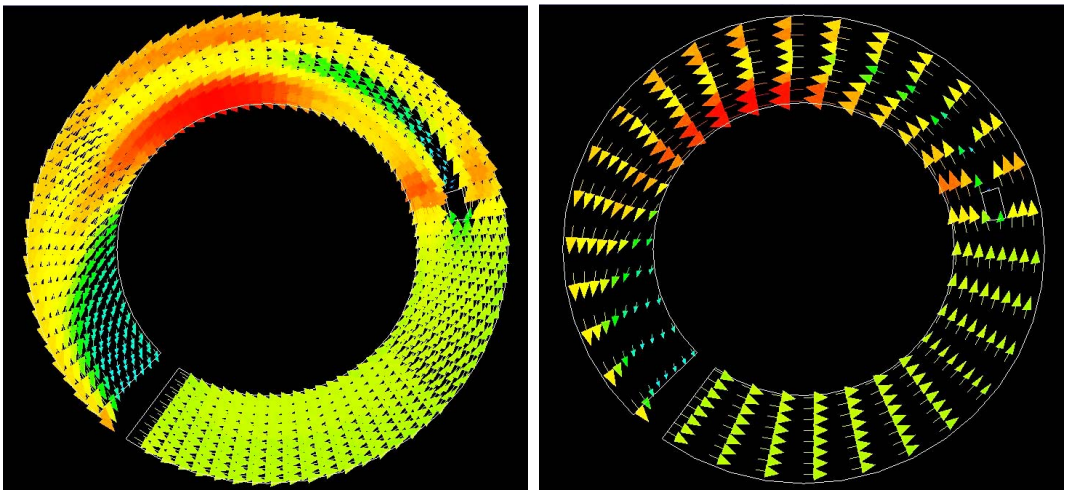


Figure 3.9: (left) Visualizing the direction of the flow through a cylindrical slice from CFD using a brute force hedgehog technique. (right) Visualizing the direction of the flow, including the normal component, using the polar resampling option of FIRST.

with several frames per second. We note that in our application a resampling grid of 10,000 cells is not practical due to perceptual problems as well as pixel resolution limits. In this case vector glyphs typically cover only one or two pixels. Also, a resampling grid with such a high resolution defeats one purpose for which it was intended, to provide a structured *summary* of the underlying data. This grid size is evaluated here in order to show the upper limits of the resampling algorithm. Grid sizes between 10×10 (like that of Figure 3.8 on page 43) and 50×50 are much more likely in our case. In practice we find that rendering the geometry is often the performance bottleneck.

Another result of the resampling algorithm is the ability to visualize unsteady flow with the normal component of the slice through which the flow is passing. In our review of the flow visualization literature [116, 117], we see several flow visualization solutions for 2D, steady-state, vector fields. However, we see no unsteady solutions that include all three vector field components at *interactive* frame rates. In the case of Scheuermann et. al. [132], theirs is a visualization solution for steady-state flow, i.e., instantaneous flow from one time step. In our case, we visualize the flow, with normal component, over time, also at interactive frame rates.

We have implemented an animation control that allows the user to load several time steps of CFD simulation result data, and "play them back" at user-specified time intervals. The user is allowed to pause, rewind, forward, and stop the animation using controls similar to any home media player. We combine this animation control with the resampling feature for an intuitive visualization of unsteady flow with normal components.

3.6 Discussion and Future Work

Our experience shows FIRST to be a valuable asset in the engineer's pursuit of understanding the underlying flow field in their CFD simulation models. FIRST reduces perceptual problems such as occlusion and visual complexity when visualizing a vector field, steady or unsteady, with all three vector field components. Searching for flow features by sliding a slice through the 3D volume is also hastened.

Future work can take multiple directions including the addition of more user-interaction controls, extending the algorithm to 3D, and porting the algorithm to Java for inclusion into the VisAD open source,

number of polygons	resolution of resampler cells	frames per second
1,552	5×5	10.0-11.0
	10×10	5.0-6.0
	50×50	1.0-6.0
	100×100	0.2-0.3
3,157	5×5	6.0-7.0
	10×10	5.0-6.0
	50×50	1.0-3.0
	100×100	0.2-1.0
14,085	5×5	1.0-2.0
	10×10	0.7-1.3
	50×50	0.17-0.30
	100×100	0.07-0.10

Table 3.1: Sample frame rates for the resampling algorithm. Performance was evaluated on a machine running Red Hat Linux 7.2 with a 1 GHz Pentium III dual processor and 512 Mbytes of RAM. Note that the frame rate also varies as a result of caching.

scientific visualization library [56].

Future work includes the addition of more user-interaction techniques. One such technique involves the addition of handles attached to the resampling grid. The user can click-and-drag a handle and change the location or size of the resampling grid. Two different strategies are possible: (1) where the other corners of the resampling grid are held fixed while one handle is dragged resulting in a change of size or (2) where the other corners move with the grid retaining a constant size but specifying a new location.

One of goals is to extend the algorithm to handle 3D simulation data. We believe that the algorithm outlined in Section 3.4 on page 38 can be extended to 3D in a very straightforward manner. The only difference would be a modification to the traversal part of the algorithm to include multiple layers. We realize, however, that performance time will be biggest challenge in realizing this goal.

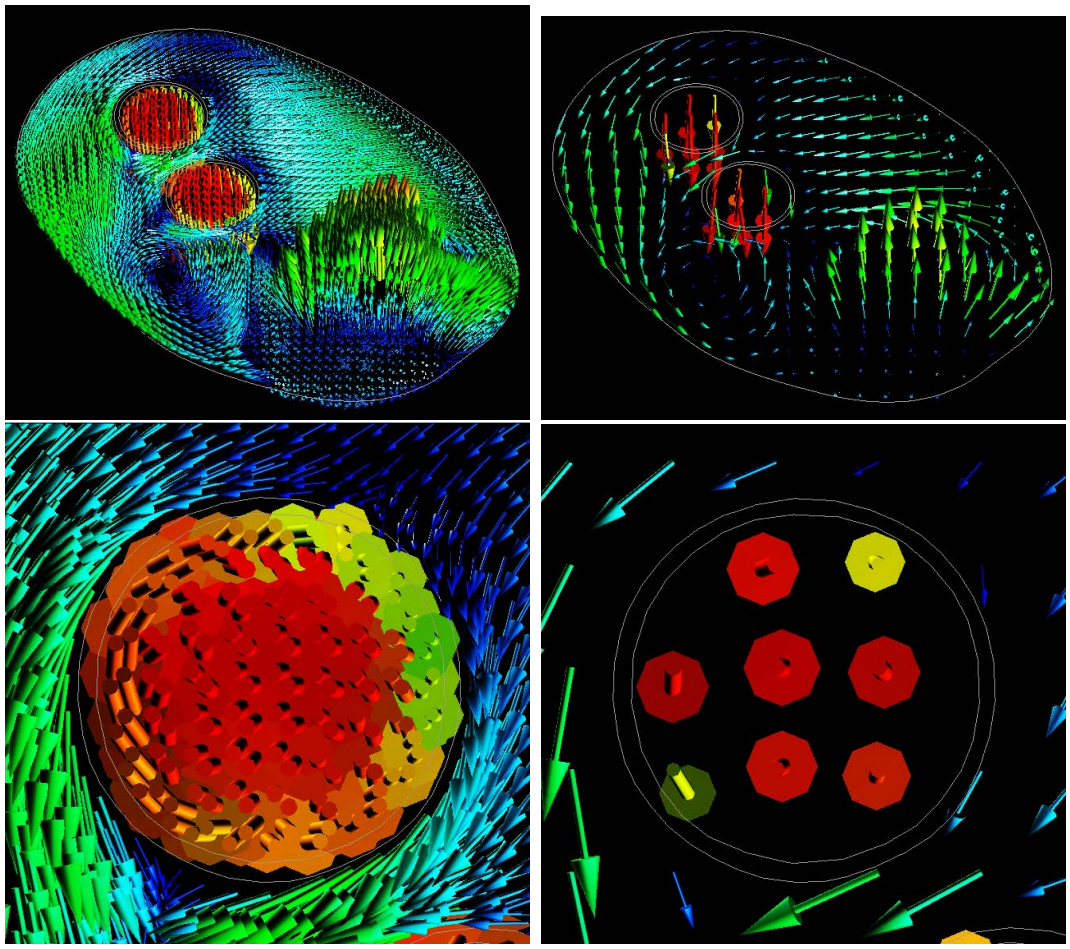


Figure 3.10: (top, left) A slice mesh with discontinuities -two gaps in the shape of rings, using hedgehog visualization (top, right) The same slice resampled onto a regular grid, (bottom, left) a close-up view of one of the rings causing a discontinuity, with hedgehog visualization (bottom, right) the same close-up view with resampling enabled

Chapter 4

ISA: Image Space Based Visualization of Unsteady Flow on Surfaces

"Speed is irrelevant if you're traveling in the wrong direction."

– Mahandas Gandhi [1] (1869–1948)

Dense, texture-based, unsteady flow visualization on surfaces has remained an elusive problem since the introduction of texture-based flow visualization algorithms themselves. The class of fluid flow visualization techniques that generate dense representations based on textures started with the Spot Noise [163] and LIC [17]. The main advantage of this class of algorithms is their *complete* depiction of the flow field while their primary drawback is, in general, the computational time required to generate the results.

Recently, two new algorithms, namely Lagrangian-Eulerian Advection (LEA) [63] and Image Based Flow Visualization (IBFV) [164], have been introduced that overcome the computation time hurdle by generating two-dimensional flow visualization at interactive frame rates, even for unsteady flow. This paves the way for the introduction of new algorithms that overcome the same problems on boundary surfaces and in three dimensions. In this chapter (which has also been published elsewhere [83]) we present a new algorithm, ISA (Image Space Advection), that generates dense representations of arbitrary fluid flow on complex, non-parameterized surfaces, more specifically, surfaces from computational fluid dynamics (CFD). However, the algorithm is general enough to apply to other vector field data associated with a surface such as blood vessel flow.

Traditional visualization of boundary flow using texture mapping first maps one or more 2D textures to a surface geometry defined in 3D space. The textured geometry is then rendered to image space. Here, we alter the classic order of operations. First we project the surface geometry to image space and then apply texturing. In other words, conceptually texture properties are advected on boundary surfaces in 3D but in fact our algorithm realizes texture advection solely in image space. The result is a versatile visualization technique with the following characteristics:

- generates a dense representation of unsteady flow on surfaces
- visualizes flow on complex surfaces composed of polygons whose number is on the order of 200,000 or more
- visualizes flow on dynamic meshes with time-dependent geometry and topology
- visualizes flow independent of the surface mesh's complexity and resolution

[1] Indian ascetic and nationalist leader, assassinated

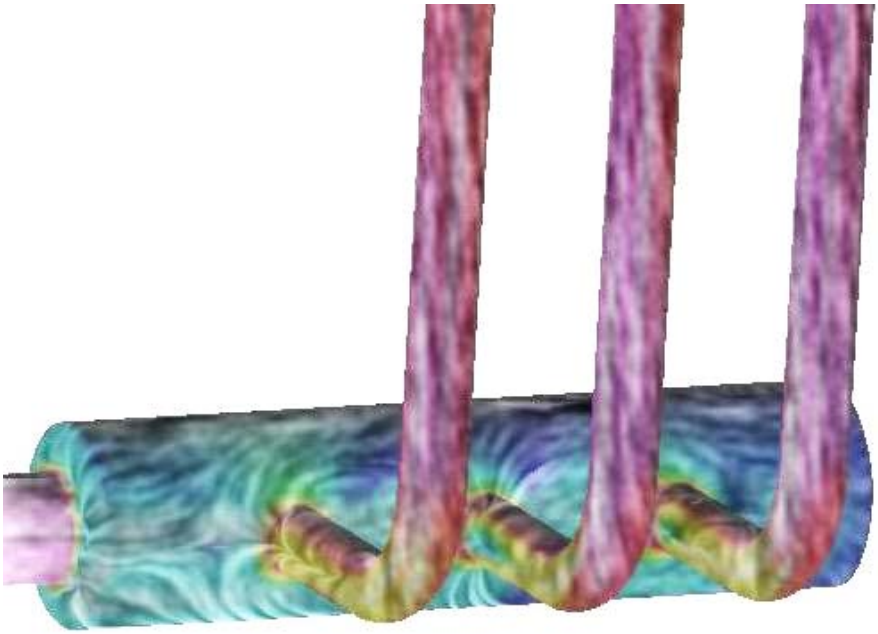


Figure 4.1: Visualization of flow on the surface of an intake manifold. The ideal intake manifold distributes flow evenly to the piston valves.

- supports user-interaction such as rotation, translation, and zooming always maintaining a constant, high spatial resolution
- the technique is fast, realizing up to 20 frames per second

The performance is due, among other reasons, to the exploitation of graphics hardware features and utilization of frame-to-frame coherency. The rest of the chapter is organized as follows: in Section 2 we discussed related work, Section 4.2 details unsteady flow visualization on surfaces from CFD. Implementation details are described in Section 4.8 while results and conclusions are discussed in Section 3.5.

4.1 Physical Space vs. Parameter Space vs. Image Space

One approach to advecting texture properties on surfaces is via the use of a parameterization, a topic that has been studied *ad nauseam* (e.g., Levy et al. [87]). According to Stalling [143], applying LIC to surfaces becomes particularly easy when the *whole* surface can be parameterized globally in two dimensions, e.g., in the manner of Forssell and Cohen [43, 44]. However, there are drawbacks to this approach. Texture distortions are introduced by the mapping between parameter space and physical space and, more importantly, for a large number of surfaces, no global parameterization is available such as isosurfaces from marching cubes and most unstructured surface meshes resulting from CFD. Surface meshes from CFD may consist of smoothly joined parametric patches, but can have a complex topology and therefore, in general, cannot be parameterized globally. Figures 4.2 and 4.3 are examples of surfaces for which a global parameterization is not easily derived.

Another approach to advecting texture properties on surfaces would be to immerse the mesh into a 3D texture, then the texture properties could be advected directly according to the 3D vector field. This would have the advantages of simplifying the mapping between texture and physical space and would

Figure 4.2: Visualization of flow at the complex surface of a cooling jacket -a composite of over 250,000 polygons.

result in no distortion of the texture. However, this visualization would be limited to the maximum resolution of the 3D texture, thus causing problems with zooming. Also, this approach would not be very efficient in that most of the texels are not used. The amount of texture memory required would also exceed that available on our graphics card, e.g., we would need approximately 500MB of texture memory if we use 4 bytes per texel and a 512^3 resolution texture.

Can the problem be reduced to two dimensions? The surface patches can be packed into texture space via a triangle packing algorithm in the manner described by Stalling [143]. However, the packing problem becomes complex since our CFD meshes are composed of many scalene triangles as opposed to the equilateral and isosceles triangles often found in computational geometry. The problem of packing scalene triangles has been studied by Carr et al. [19]. For CFD meshes, triangles generally have very disparate sizes. For a given texture resolution, many triangles would have to be packed that cover less than one texel. To by-pass this, the surfaces could be divided into several patches which could be stored into a texture atlas [87]. In any case, computation time would be spent generating texels which cover polygons hidden from the current point of view. The preceding discussion leads us to an alternative solution that, ideally, has the following characteristics: works in image space, efficiently handles large numbers of surface polygons, spends no extra computation time on occluded polygons, does not spend computation time on polygons covering less than a pixel, and supports user interaction such as zooming, translation, and rotation.

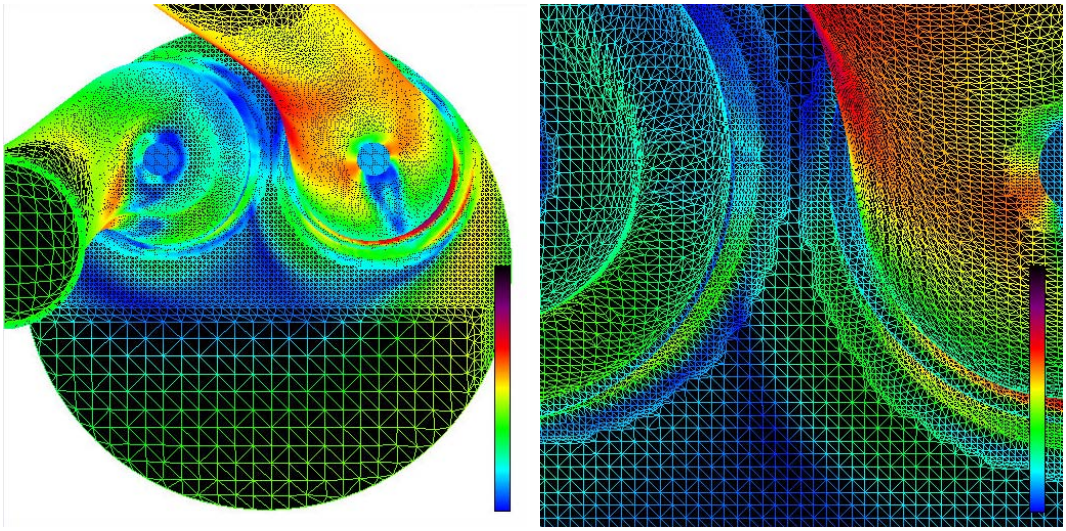


Figure 4.3: A wire frame view of the surface of two intake ports showing its 221,000 polygonal composition: (left) an overview from the top, note that many polygons are cover less than one pixel (right) a close-up view of the mesh between the two intake ports.

4.2 Method Overview

The algorithm presented here simplifies the problem by confining the advection of texture properties to image space. We project the surface geometry to image space and then apply a series of textures. This order of operations eliminates portions of the surface hidden from the viewer. In short, our proposed method for visualization of flow on surfaces is comprised of the following procedure:

1. associate the 3D flow data with the polygons at the boundary surface i.e., a velocity vector is stored at each polygon vertex of the surface
2. project the surface and its vector field onto the image plane
3. identify geometric discontinuities
4. advect texture properties according to the vector field in image space
5. inject and blend noise
6. apply additional blending along the geometric discontinuities previously identified
7. overlay all optional visualization cues such as showing a semi-transparent representation of the surface with shading

These stages are depicted schematically in Figure 4.4. Each step of the pipeline is necessary for the dynamic cases of unsteady flow, time-dependent geometry, rotation, translation, and scaling, and only a subset is needed for the static cases involving steady-state flow and no changes to the view-point. We consider each of these stages in more detail in the sections that follow.

4.3 Vector Field Projection

In order to advect texture properties in image space, we must project the vector field associated with the surface to the image plane, taking into account that the velocity vectors are stored at the polygon vertices. We chose to take advantage of the graphics hardware to project the vector field to the image plane. We assign a color whose R, G, and B values encode the x, y, and z components of the local vectors to each vertex of the boundary surface respectively. The velocity-colored geometry is rendered

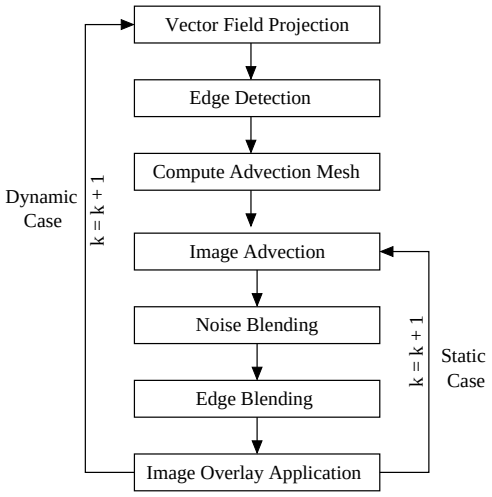


Figure 4.4: Flow diagram of texture-based flow visualization on complex surfaces -k represents time as a frame number.

to the framebuffer. We use the term *velocity image* to describe the result of encoding the velocity vectors at the mesh vertices into color values. The velocity image is interpreted as the vector field and is used for the texture advection in image space. More precisely, the color assignment can be done with a simple scaling operation. For each color component, $\mathbf{h}_{rgb}$, we assign a velocity, $\mathbf{v}_{xyz}$ component according to:

$$\begin{aligned} \mathbf{h}_r &= \frac{\mathbf{v}_x - \mathbf{v}_{min\,x}}{\mathbf{v}_{max\,x} - \mathbf{v}_{min\,x}} \\ \mathbf{h}_g &= \frac{\mathbf{v}_y - \mathbf{v}_{min\,y}}{\mathbf{v}_{max\,y} - \mathbf{v}_{min\,y}} \\ \mathbf{h}_b &= \frac{\mathbf{v}_z - \mathbf{v}_{min\,z}}{\mathbf{v}_{max\,z} - \mathbf{v}_{min\,z}} \end{aligned} \tag{4.1}$$

The minimum velocity component is subtracted for each color component respectively, in an effort to minimize loss of accuracy.

The use of a velocity image yields the following benefits: (1) the advection computation and noise blending is simpler in image space, thus we inherit advantages from the LEA and IBFV, (2) the vector field and polygon mesh are decoupled, thereby freeing up expensive computation time dedicated to polygons smaller than a single pixel, (3) conceptually, this is performing hardware-accelerated occlusion culling, since all polygons hidden from the viewer, are immediately eliminated from any further processing, and (4) this operation is supported by the graphics hardware. Saving the velocity image to main memory is simple, fast, and easy. A sample velocity image is shown in Figure 4.5 (top, left).

The construction of the velocity image allows us to take advantage of hardware-accelerated flow field reconstruction. During the construction of the velocity image, we enable Gouraud Shading, also supported by the graphics hardware. Since each velocity component is stored as hue at each polygon vertex of the surface, the smooth interpolation of hue amounts to hardware-accelerated vector field reconstruction. This is important for a minimum of two reasons. First, the polygonal primitive we choose at image advection time is independent of the original mesh polygons (more in Section 4.4). In other words, the vertices of the mesh we use to distort the image are not the same vertices where the original velocity vectors are stored. Second, interpolation is essential for flow field reconstruction. When the surface is rendered with velocity encoded as hue, the vertices of some polygons are clipped during the projection

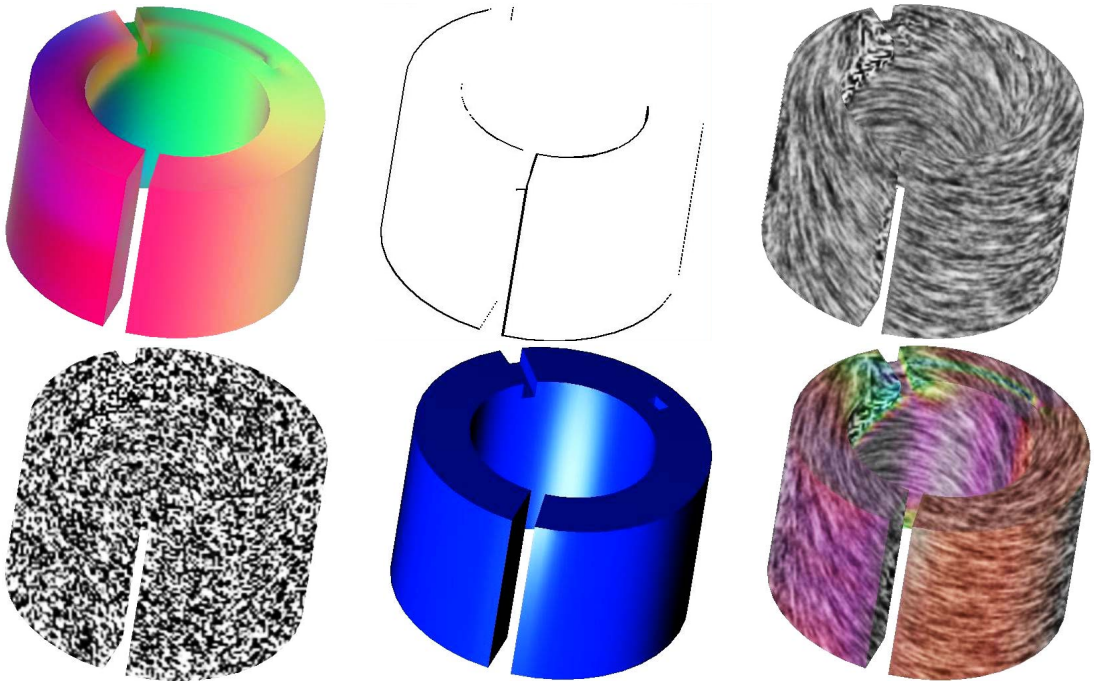


Figure 4.5: The 5 component images, plus a 6^{th} composite image, used for the visualization of surface flow on a ring: (top, left) the velocity image, (top, middle) the geometric edge boundaries, (top, right) the advected and blended textures, (bottom, left) a sample noise image, (bottom, middle) an image overlay, (bottom, right) the result of the composited images with an optional velocity color map. The geometric edge boundaries are drawn in black for illustration.

process. However, we still need to access the vector field values inside those polygons, and not just at the vertices, hence the need for reconstruction. We also note that we are not necessarily limited to linear interpolation for reconstruction. Higher order interpolation schemes can be supported by graphics hardware [50].

The velocity vectors are de-coded from the velocity image according to:

$$\begin{aligned} \mathbf{v}_x &= \mathbf{h}_r \cdot (\mathbf{v}_{max\,x} - \mathbf{v}_{min\,x}) + \mathbf{v}_{min\,x} \\ \mathbf{v}_y &= \mathbf{h}_g \cdot (\mathbf{v}_{max\,y} - \mathbf{v}_{min\,y}) + \mathbf{v}_{min\,y} \\ \mathbf{v}_z &= \mathbf{h}_b \cdot (\mathbf{v}_{max\,z} - \mathbf{v}_{min\,z}) + \mathbf{v}_{min\,z} \end{aligned} \tag{4.2}$$

The de-coded velocity vectors used to compute the advection mesh (Sec 4.4) are then projected onto the image plane.

The magnitude of the velocity vectors at those parts of the surface orthogonal to the image plane may be shortened as a result of perspective projection, i.e., if the dot product between the image plane normal and the 3D surface normal is zero or close to zero. This can reduce the visual clarity of the vector field's direction during animation. In our implementation, we added an option that allows the user to apply a bias to the velocity vectors that are shortened close to zero due to the projection. We can use this bias to reduce the scaling effect for curved surfaces. Conceptually it is like applying a reverse velocity clamp.

The projection of the vectors to the image plane is done after velocity image construction for 2 reasons: (1) not *all* of the vectors have to be projected (Sec. 4.4), thus saving computation time and (2) we use the original 3D vectors for the velocity mask (Sec. 4.8).

4.4 Advection Mesh Computation and Boundary Treatment

After the projection of the vector field we compute the mesh used to advect the textures similar to IBFV. We distort a regular, rectilinear mesh according to the velocity vectors stored at mesh grid intersections. The distorted mesh vertices can then be computed by advecting each mesh grid intersection according to the discretized Euler approximation of a pathline, $\mathbf{p}$, (the same as a streamline for steady flow) expressed as:

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \mathbf{v}^p(\mathbf{p}_k; t)\,\Delta\, t \tag{4.3}$$

where $\mathbf{v}^p$ represents a magnitude and direction after projection to the image plane. The texture coordinates located at the regular, rectilinear mesh vertices are then mapped to the (forward) distorted mesh positions. The distorted mesh positions are stored for fast advection of texture properties for static scenes.

Special attention must be paid in order to handle flow at geometric boundaries of the surface. Figure 4.6 shows an overview of the original IBFV process. During the visualization, each frame is advected, rendered, and blended in with a background image. If we look carefully at the *distort* phase of the algorithm, we notice that there is nothing to stop the image from being advected outside of the physical boundary of the geometry. While this is not a problem when the geometry covers the entire screen, this can lead to artifacts for geometries from CFD, especially in the case of boundaries with a non-zero outbound flow, e.g., flow outlets.

To address this problem we borrow a notion from LEA that treats non-rectangular flow domains, namely, the use of backward coordinate integration (also proposed by Max and Becker [103]). Using backward integration, equation 4.3 becomes:

$$\mathbf{p}_{k-1} = \mathbf{p}_k - \mathbf{v}^p(\mathbf{p}_{k-1}; t)\,\Delta\, t \tag{4.4}$$

In this case the texture coordinates located at the (backward) distorted mesh positions are mapped to the regular, rectilinear mesh vertices. Backward integration does not allow advection of image properties past the geometric boundaries.

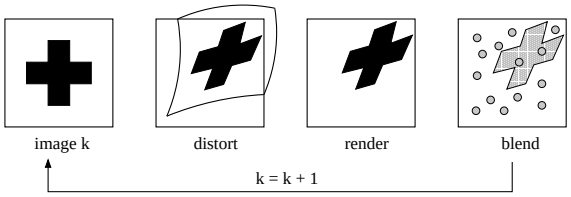


Figure 4.6: An overview of the original image based flow visualization

4.5 Edge Detection and Blending

While we gain many advantages by decoupling the image advection process with the 3D surface geometry, artifacts can result, especially in the case of geometries with sharp edges. If we look carefully at the result of advecting texture properties in image space, we notice that in some cases a visual flow continuity is introduced where it may be undesirable. A sample case is shown in Figure 4.7. A portion of the 3D geometry, shown colored, is much less visible after the projection onto the image plane. If the flow texture properties are advected across this edge in image space, also shown colored, an artificial continuity results. To handle this, we incorporate a geometric edge detection process into the algorithm.

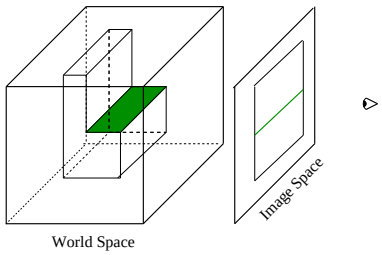


Figure 4.7: When a 3D surface geometry (left) is projected, continuity is created in image space (right). If the flow aligned texture properties are advected across this edge, an artificial flow continuity may result.

During the image integration computation, we compare spatially adjacent depth values during one integration and advection step. We compare the associated depth values, z_{k-1} and z_k in world space of $\mathbf{p}_{k-1}$ and $\mathbf{p}_k$ from equation 4.4, respectively. If

$$|z_{k-1} - z_k| > \varepsilon \cdot |\mathbf{p}_{k-1} - \mathbf{p}_k| \tag{4.5}$$

where ε is a threshold value, then we identify an edge. All positions, $\mathbf{p}$, for which equation 4.5 is true, are classified as edge crossing start points. Special treatment must be given when advecting texture properties from these points. This process does not detect *all* geometric edges, only those edges across which flow texture properties should not be advected.

Figure 4.5 top, middle shows one set of edges from the detection process. The geometric edges are identified and stored during the dynamic visualization case and additional blending is applied (depicted schematically in Figure 4.4). During the edge blending phase of the algorithm we introduce discontinuities in the texture aligned with the geometric discontinuities from the surface, i.e., gray values are blended in at the edges. This has the effect of adding a gray scale phase shift to the pixel values already blended. This could obviously be handled in different ways, e.g., choosing a random noise value to advect or inverting the noise value already present. Some results of the edge detection and blending phase are illustrated in Figure 4.8. In our data sets an ε of 1-2% of depth buffer is practical. However, the users may set their own value if fine tuning of the visualization is needed.

The same edge detection and blending benefits incoming boundary flow. Also an artifact of the IBFV algorithm, geometric boundaries with incoming flow may appear dimmer than the rest of the geometry. This is a result of the noise injection and blending process described in Section 4.6. In short, the background color shows through more in areas of incoming flow because not as much noise has been blended in these areas. Figure 4.9, top, shows a 2D slice through a 3D mesh from a CFD simulation with incoming boundary flow coming in through the narrow inlet from the right. Note that the edge of the inlet appears dim. Figure 4.9, bottom, shows the same slice with edge blending turned on. The boundary artifacts of the noise injection and blending process are no longer a distraction. Edge detection and blending also plays in important role while an object is rotating. Without special treatment, contours in image space become blurred when different portions of a surface geometry overlap, such as when blood vessels in Figure 4.11 overlap during rotation.

4.6 Noise Blending

By reducing the image generation process back to two dimensions, the noise injection and blending phase falls in line with the original IBFV technique, namely, an image, F, is related to a previous image,

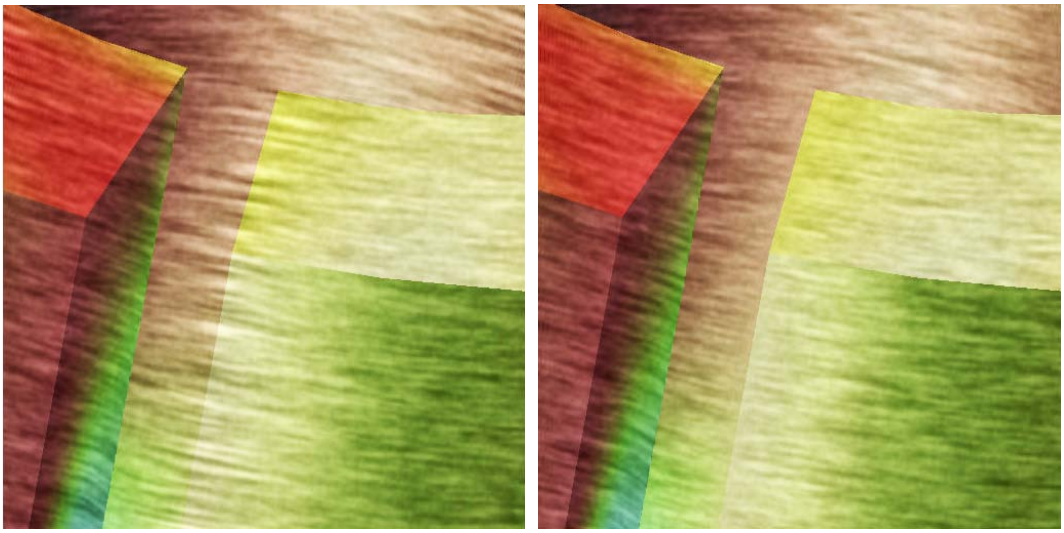


Figure 4.8: A close-up example of geometric edge detection: on the left side, geometric edge detection is disabled, on the right side enabled.

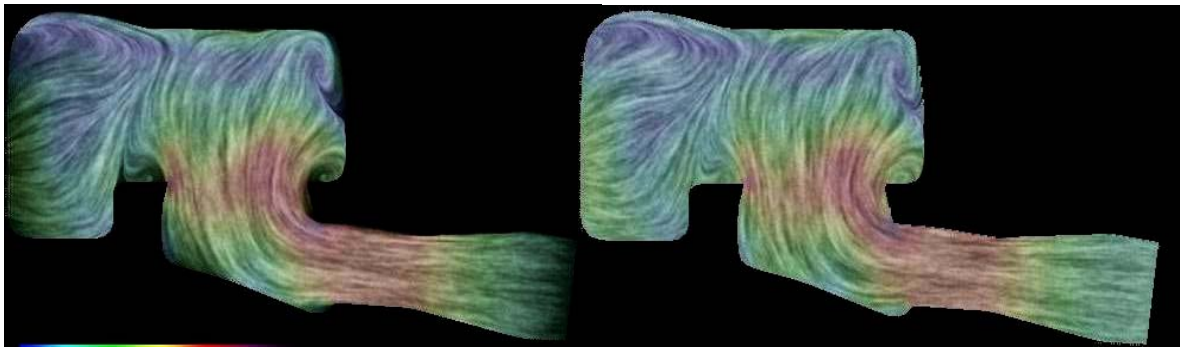


Figure 4.9: Here we see a 2D slice through a 3D geometry from a CFD simulation. (top) With no edge blending, the background color shows through boundary areas with incoming flow. (bottom). With edge blending, these artifacts are no longer a distraction.

G, by [164]:

$$F(\mathbf{p};k) = \alpha \sum_{i=0}^{k-1} (1-\alpha)^i G(\mathbf{p}_{k-i}; k-i) \tag{4.6}$$

where **p** represents a pathline, α defines a blending coefficient, and k represents time as a frame number. Thus a point, $\mathbf{p}_k$, of an image F_k, is the result of a convolution of a series of previous images, $G(\mathbf{x};i)$, along the pathline through $\mathbf{p}_k$, with an decay filter defined by $\alpha(1-\alpha)^i$. The blended noise images have both spatial and temporal characteristics. In the spatial domain, a single noise image, $g(x)$, is described as a linearly interpolated sequence of n random values, G_i, in the range $[0, n-1]$, i.e.,

$$g(x) = \sum h_s(x - is)\, G_{i \bmod n} \tag{4.7}$$

where the spacing, s, between noise samples is generally greater than or equal to the distance traversed by an image property in one advection step and h_s represents a triangular black and white pulse function. Here x represents a location in the flow domain. In practice, we give the user control of s, resulting

in multi-frequency texture resolutions in the spacial domain. The background textures used for blending also vary in time. In the temporal domain, each point, G_i in the background texture, periodically increases and decays according to a profile, $w(t)$, e.g.,

$$G_{i;k} = w((k/M + \phi_i)\, mod\, 1) \tag{4.8}$$

where ϕ_i represents a random phase, drawn from the interval [0,1), M is the total number of background noise images used, and where $w(t)$ is defined for all time steps. We use a square wave profile, i.e., $w(t) = 1$ if $t < 1/2$ and 0 otherwise. In our application, the user has the option of varying M. Smaller values of M result in higher frequency noise in the temporal domain whereas higher values M result in a lower temporal frequency. Figure 4.5 (top, right) shows a sample blended image and Figure 4.5 (bottom, left) shows a sample noise image.

4.7 Image Overlay Application

The rendering of the advected image and the noise blending may be followed by an optional image overlay. An overlay enhances the resulting texture-based representation of surface flow by applying color, shading, or any attribute mapped to color (Fig. 4.5, bottom, middle). In implementation, we generate the image overlay following the construction of the velocity image. This overlay may render any CFD simulation attribute mapped to hue. The overlay is constructed once for each static scene and applied after the image advection, edge blending, and noise blending phases. Since the image advection exploits frame-to-frame coherency, the overlay must be applied after the advection in order to prevent the surface itself from being smeared. Also worthy of mention, is that the opacity value of the image overlay is a free parameter we provide to the user.

4.8 Implementation

In this section we consider some aspects of the algorithm not previously discussed which are important for implementation. Our implementation is based on the highly portable OpenGL 1.1 (`www.opengl.org`) library.

Texture Clipping

In our application, the resolution of the quadrilateral mesh used to warp the image can be specified by the user. The user may specify a coarse resolution mesh, e.g., 128×128, for faster performance or a fine resolution mesh, e.g., 512×512, for higher accuracy. However, if the resolution of the advection mesh is too coarse in image space, artifacts begin to appear. Figure 4.10, left, illustrates these artifacts zoomed in on the edge of a surface. In order to minimize the jagged edges created by coarse resolution texture quadrilaterals, we apply a texture clipping function. Subsets of textured quadrilateral that do not cover the surface are clipped from the visualization as shown in Figure 4.10, right. This can be implemented simply with the image overlay by maximizing the opacity wherever the depth buffer value is maximized, i.e., wherever there is a great depth.

Velocity Mask

In order to dim high frequency noise in low velocity regions, the user also has the option of applying a velocity mask. We adopt the velocity mask of Jobard et al. [63] for our purposes here, namely:

$$\alpha = 1 - (1 - \mathbf{v})^m \tag{4.9}$$

where α decreases as a function of velocity magnitude. In our case, the image overlay becomes more opaque in regions of low velocity and more transparent in areas of high velocity. With the velocity mask

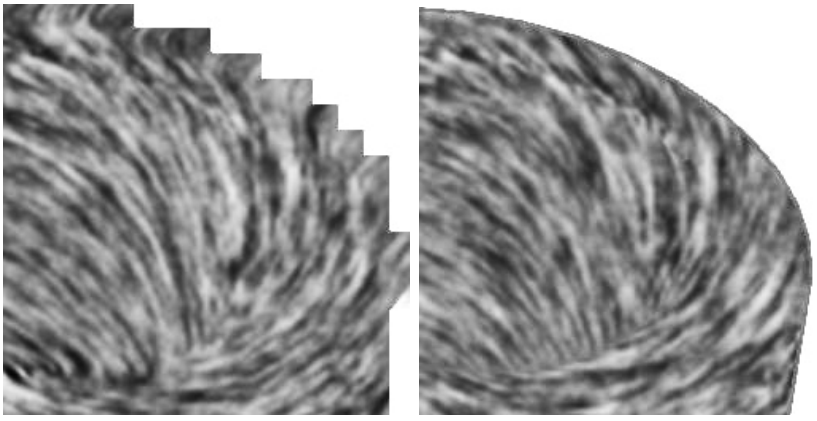


Figure 4.10: The result of, left, a coarse resolution advection mesh with artifacts and, right, the application of texture clipping. The resolution of the advection mesh shown on the left is 32×32 for illustration.

enabled, the viewer's attention is drawn away from areas of stagnant flow, and towards areas of high flow velocity. We note that in the context of CFD simulation data, engineers are often very concerned about areas of stagnant flow. In the case of a cooling jacket, stagnant flow may represent a region of the geometry where the temperature is too high, possibly leading to boiling conditions thus reducing the effectiveness of the cooling jacket itself. Therefore, in our case the engineers may disable the velocity mask or use the velocity mask to *highlight* areas of flow, e.g., make the hue brighter in areas of low velocity.

4.9 Performance and Results

Our visualization technique is applied primarily to large, highly irregular, adaptive resolution meshes commonly resulting from computational fluid dynamics simulations. [2] The ideal intake manifold (Fig. 4.1) supplies an equal amount of fluid flow to each piston valve. Visualizing the flow at the surface gives the engineer insight into any imbalances between the inlet pipes, in this case, the 3 long narrow pipes of the geometry. Figure 4.12 on page 60 shows our method applied to a surface of an intake port mesh (from Figure 4.3 on page 50) composed of 221K polygons. The intake port mesh is composed of highly adaptive resolution surface polygons and for which no global parameterization is readily available. The method described here allows the user to zoom in at arbitrary view points always maintaining a high spatial resolution visualization.

The algorithm applies equally well to meshes with time-dependent geometry and topology. Figure 4.13 on page 61 shows the surface of a piston cylinder with the piston head (not shown) defining the bottom of the surface. The method here enables the visualization of fuel intake as the piston head slides down the cylinder. The resulting flow visualization has a smooth spatio-temporal coherency. Our algorithm also has applications in the field of medicine. Figure 4.11 on page 58 shows the circulation of blood at the junction of 3 blood vessels. An abnormal cavity has developed that may hinder the optimal distribution of blood.

Performance was evaluated on an HP Visualize workstation with an HP *fx* graphics card, running Red Hat Linux 7.2 with a 1 GHz Pentium III dual processor and 1 GB of RAM. The performance times reported in Table 4.1 on page 59 support interactive exploration of unsteady flow on surfaces. The first

[2] Supplementary video available at: `http://www.VRVis.at/ar3/pr2/vis03/`

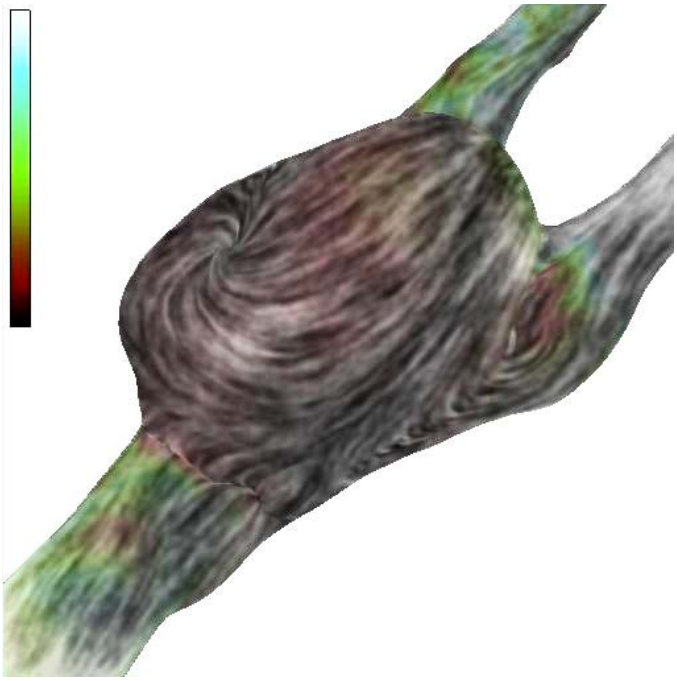


Figure 4.11: Visualization of blood flow at the surface of the junction of 3 blood vessels. Stagnant blood flow may occur within the abnormal pocket at the junction.

time reported in the FPS column is for the static cases of steady-state visualization and the absence of changes to the view point. The times shown in parenthesis indicate the dynamic cases of unsteady flow and interactive zooming and rotation. More specifically, the dynamic cases require the construction of a velocity image, image overlay, as well as geometric edge detection. We include geometric edge detection in the frame rates reported in Table 4.1 on page 59. It does not introduce significant overhead since it is easily built into the advection process itself.

The performance time of our algorithm depends on the resolution of the mesh used to perform the advection and the number of polygons in the original surface mesh. In the case of steady-state flow, the algorithm no longer depends on the number of polygons in the surface mesh, but on the area covered in image space. The data set shown in Figure 4.1, left, on page 48, does not cover as much image space, so its performance times are somewhat higher in the static case. A full discussion of advection mesh resolution and image quality is given in a comparison of ISA and IBFVS [85].

4.10 Discussion and Future Work

We have presented a novel technique for dense representations of unsteady flow on boundary surfaces from CFD. The algorithm supports visualization of flow on arbitrary surfaces at up to 20 FPS via the careful use of graphics hardware. It supports exploration and visualization of flow on large, unstructured polygonal meshes, and on time-dependent meshes with dynamic geometry and topology. The method generates dense representations of time-dependent vector fields building on both the LEA and IBFV algorithms. It also does not waste computation time on occluded polygons or polygons covering less than one pixel. While the vector fields are defined in 3D and associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space.

data set	number of polygons	advection mesh resolution	frames per second
ring (Fig 4.5)	10K	128×128	18 (5)
		256×256	9 (3)
		512×512	3 (1)
intake manifold (Fig 4.1)	48K	128×128	22 (2)
		256×256	14 (2)
		512×512	6 (1)
combustion chamber (Fig 4.13)	79K	128×128	17 (2)
		256×256	10 (2)
		512×512	4 (1)
intake port (Fig 4.12)	221K	128×128	17 (0.5)
		256×256	7 (0.5)
		512×512	2 (0.3)

Table 4.1: Sample frame rates for the visualization algorithm.

Future work can go in many directions including visualization of unsteady 3D flow, something we expect to see soon. Challenges will include both interactive performance time and perceptual issues. Future work also includes the application of more specialized graphics hardware features like programmable per-pixel operations in the manner of Weiskopf et al. [179, 181] and the use of pixel textures like Heidrich et al. [52].

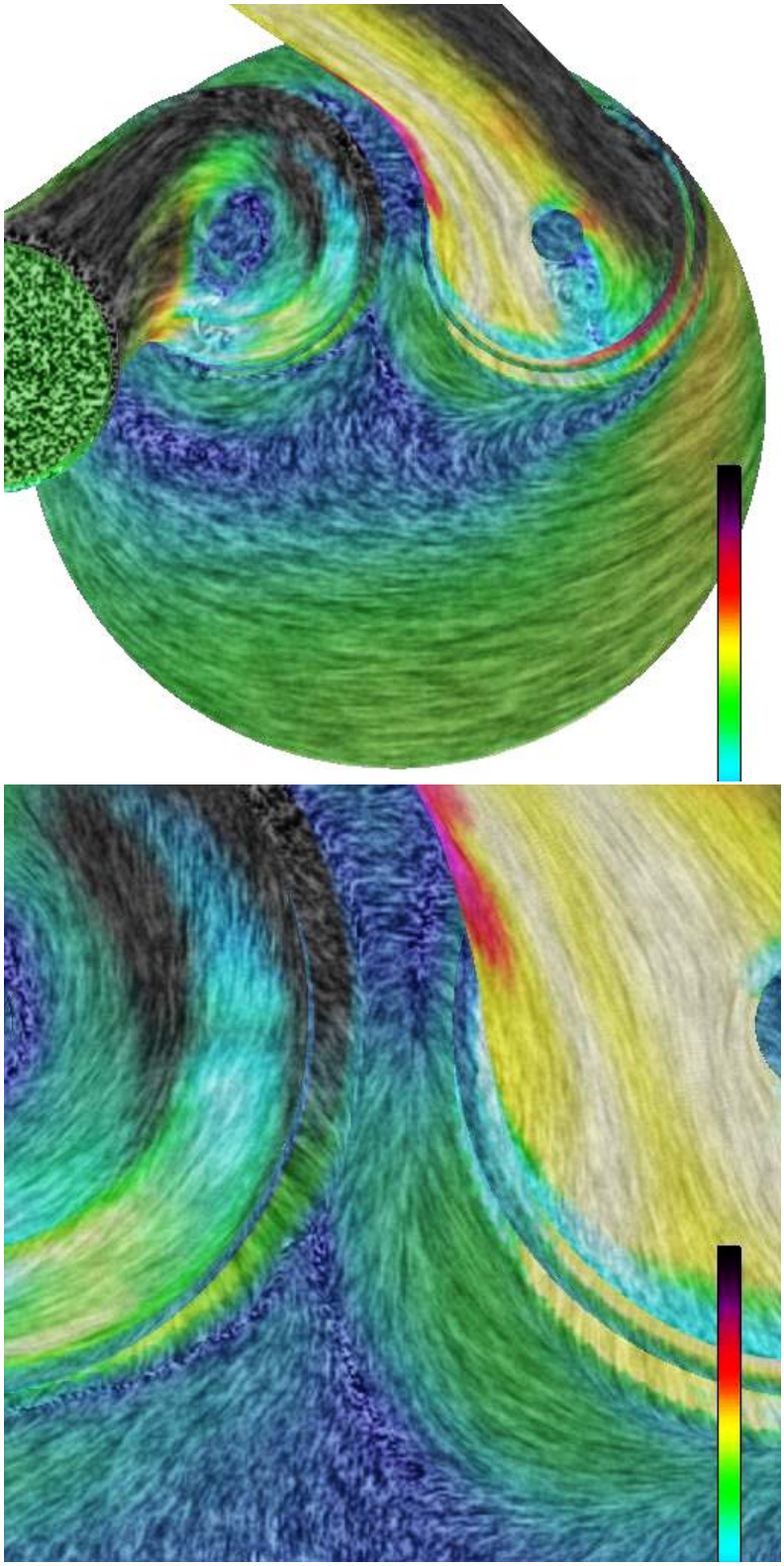


Figure 4.12: (top) A view of the surface of an 221K polygonal intake port mesh show in Figure 4.3 on page 50. Texture-based flow visualization is applied to the surface. (bottom) A close-up view of the surface of the intake port mesh. Here we illustrate user-supported zooming with automatic, on the fly recalculation of the flow texture.

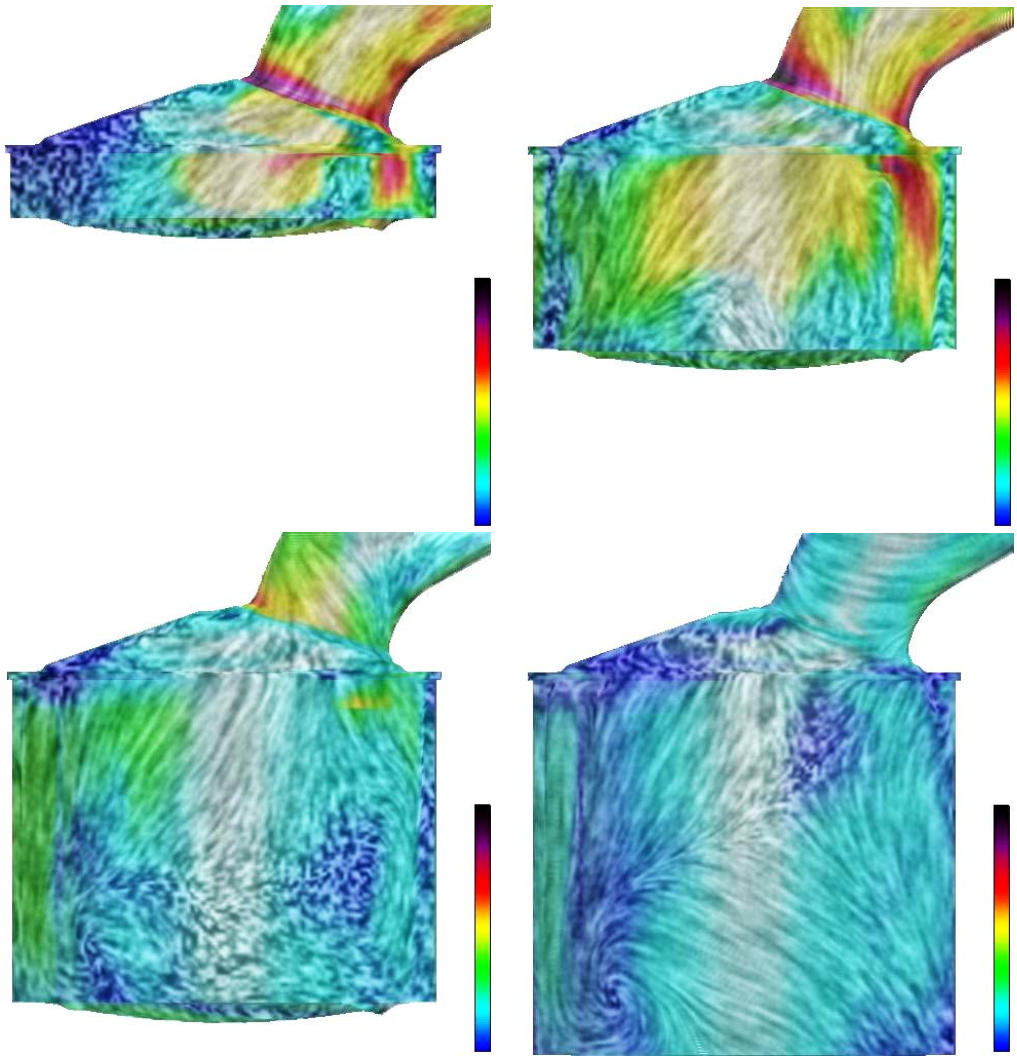


Figure 4.13: Snapshots from the visualization of a time-dependent surface mesh composed of a 79K polygons with dynamic geometry and topology. This intake valve and piston cylinder can also be used to analyse the formation of *wall film*, the term used to describe the liquid buildup on surfaces.

Chapter 5

Texture-Based Flow Visualization on Isosurfaces from Computational Fluid Dynamics

"Men occasionally stumble over the truth, but most of them pick themselves up and hurry off as if nothing ever happened."

– Sir Winston Churchill [1] (1874–1965)

At the VRVis Research Center we collaborate with AVL (`www.avl.com`) in order to provide visualization solutions for analysis of their CFD simulation result data. AVL's own engineers as well as engineers at industry affiliates use visualization software to analyze and evaluate the results of their automotive design and simulation.

For many of the automotive components that undergo evaluation, there is an ideal pattern of flow the engineers are trying to create. Figure 5.1 illustrates the swirl motion of fluid flow in a combustion chamber from a diesel engine. In order to generate swirl motion, fluid enters the combustion chamber from the intake ports. Later on in the engine cycle, the kinetic energy associated with this swirl motion is used to generate turbulence for mixing of fresh oxygen into the fluid. The more turbulence generated, the better the mixture of air and diesel fuel, and thus the better the combustion itself. Ideally, enough turbulent mixing is generated such that 100% of the fuel is burned.

Since it is the swirling flow that is used to generate turbulence, the swirl should be maximized in order to maximize turbulence. From the point of view of the mechanical engineers designing the intake ports, increased swirl flow leads to some beneficial conditions: (1) improved mixture preparation, i.e., more fuel contact with oxygen, (2) a higher EGR (Exhaust Gas Ratio) which means a decrease in fuel consumption, and (3) lower emissions. However, too much swirl displaces the flame used to ignite the fuel. As such, a balance must be achieved between (1) generating enough swirl flow in order to create turbulence and (2) not displacing the flame used to ignite the flow.

Some routine questions that a mechanical engineer may ask when investigating swirl flow are: (1) Can visualization provide insight into or verify the characteristic shape(s) or behavior of the flow? (2) What

[1] British politician, British first lord of the admiralty 1911–1915, 1939–1940, British prime minister 1940–1945, 1951–1955, Nobel Prize in literature 1953, made honorary US citizen 1963

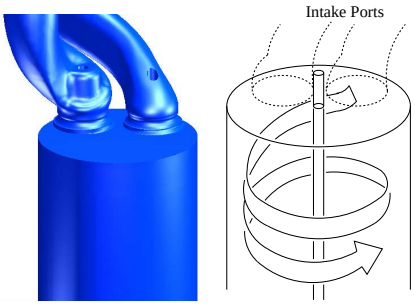


Figure 5.1: The swirling motion of flow in the combustion chamber of a diesel engine (side view). The intake ports at the top provide the tangential component of the flow necessary for swirl.

tool(s) can help to visualize the swirl flow pattern? and (3) Where in the combustion chamber is the swirl flow pattern *not* being met?

5.1 Isosurfaces

Engineers often start an analysis of CFD simulation data using techniques to visualize the flow at the surface in order to get a first impression of the simulation results. The next logical step is to investigate the properties of the flow *inside* the volume. Slices are commonly used, but visualizing 3D characteristics of the flow like swirl can be difficult with 2D slices. Engineers are interested in visualization techniques that provide insight into the spatial dimension orthogonal to the slice as well.

Isosurfaces are a visualization tool used routinely by mechanical engineers to investigate the properties of the flow inside a 3D volume. The shape of an isosurface can give the engineer insight into its 3D characteristics. One reason engineers use isosurfaces, as opposed to say streamsurfaces, is because they are so ubiquitous. They feel very comfortable with isosurfaces because, like isolines, they are very familiar. The mechanical engineers we spoke to are not as familiar with the notion of a streamsurface and even less its interpretation.

Figure 5.2 shows a velocity isosurface in the combustion chamber of the data set in Figure 5.1. The engineer can see that the flow has some of the swirling orientation that they are looking for. However, what is missing from Figure 5.2 is a clear indication of flow direction, e.g., the upstream and downstream nature of the flow. In particular, it is not obvious where the flow does *not* follow the ideal swirl pattern that the combustion chamber should encapsulate.

5.2 Applying Texture-Based Flow Visualization

Applying texture-based flow visualization techniques to such isosurfaces provides engineers even more insight into the characteristics of 3D vector fields. And this has become a feasible option only recently. We apply the ISA method [83] from Chapter 4 for producing dense, texture-based representations of flow on isosurfaces. The result is a combination of two well known scientific visualization techniques, namely iso-surfacing and texture-based flow visualization, into a useful hybrid approach. Our application is a versatile visualization technique with the following characteristics: (1) generates a dense representation of flow on adaptive resolution isosurfaces, (2) visualizes flow on complex isosurfaces composed of polygons whose number is on the order of 200,000 or more, (3) visualizes flow independent of the isosurface mesh's complexity and resolution, (4) supports user-interaction such as rotation, translation, and zooming always maintaining a constant, high spatial resolution, and (5) produces fast animations, realizing up to 60 frames per second.

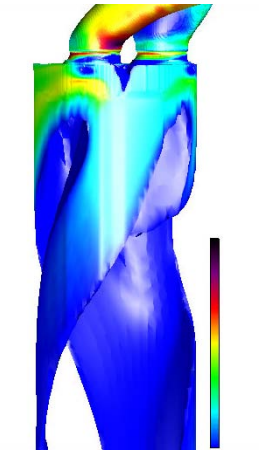


Figure 5.2: The swirling motion of flow in the combustion chamber of a diesel engine (side view) as illustrated by an isosurface of velocity. This is a velocity isosurface with an isovalue of 5.0 m/s. Any CFD attribute can be mapped to hue.

We note that the method must be applicable to adaptive resolution isosurfaces like that of Figure 5.3. Note that many of the polygons in Figure 5.3 cover less than one pixel. The isosurface algorithm used here is an extension of the Marching Cubes [97] and Marching Tetrahedra [156] algorithms that takes into account more cell types. It handles adaptive resolution meshes in the same spirit as Laramee and Bergeron [79].

The rest of the chapter is organized as follows: in Chapter 2 we discussed related work, Section 5.3 reviews the research that the work here is built upon. Section 5.4 details texture-based flow visualization on isosurfaces from CFD. Results and conclusions, including a discussion of the user questions, are presented in Section 5.5. We note that this chapter has also been published elsewhere [84].

5.3 Method Background

In order to understand how and why we apply texture-based flow visualization to isosurfaces, we briefly outline the method background. In brief, the algorithm presented by Laramee et al. [83] simplifies the problem by confining the advection of texture properties to image space. The surface geometry is projected to image space and then a series of textures are mapped, blended, and advected. This order of operations eliminates portions of the surface hidden from the viewer. The previous method for visualization of flow on surfaces is comprised of the following procedure: (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. Steps 1-7 of the pipeline are necessary for the dynamic cases of changes to the isovalue, 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 or isovalue. Each stage is described in more detail in previous research [83] and in chapter 4.

5.4 Texture-Based Flow Visualization on Isosurfaces

Here we describe the way in which we apply the method described in Section 5.3 to isosurfaces. Specifically, we describe ways to address: (1) the normal component of the flow to the isosurface, (2) perceptual challenges associated with viewing flow on isosurfaces, (3) issues related to resampling the vector field, and (4) some implementation details.

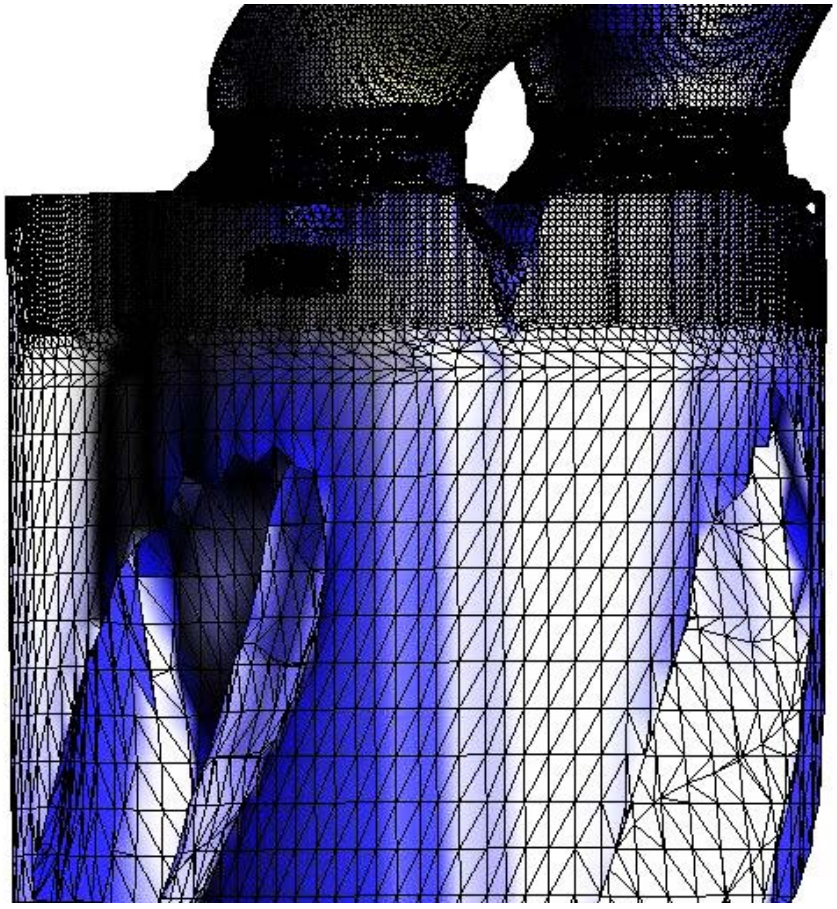


Figure 5.3: A close-up, wire-frame view of the isosurface from Figure 5.2. The algorithm we describe must be applicable to adaptive resolution isosurface meshes.

Applying a Normal Mask

When visualizing flow on normal boundary surfaces the direction of the flow generally coincides with the surface itself. As the flow approaches the boundary, it is not allowed to pass through and is pushed in a tangential direction, i.e., it can be described as surface aligned flow. However, in the case of isosurfaces this is no longer true. The flow at an isosurface can sometimes exhibit a strong flow that is normal to the surface, e.g., cross-surface flow. The same also holds true for the case of arbitrary clipping geometries. Simply advecting texture properties according to the vector field projected onto the isosurface could be considered misleading. Is there a way in which this cross-surface component of the flow can be incorporated into the result visualization?

Battke et al [3], who applied LIC to surfaces, address this problem by varying the length of the convolution filter according to the magnitude of the vector component tangential to the surface. In areas where the vector field is oriented almost perpendicular to the surface only very little smearing of the texture occurs, i.e., the input noise is visible instead of a convolved texture. Our approach is required to be consistent with the visualization of flow on boundary surfaces. When we apply texture-based flow visualization to boundary surfaces, the amount of texture-smearing indicates velocity magnitude, i.e., texture is smeared into longer streaks in areas of higher velocity magnitude. We don't want to change the semantic interpretation of smearing for isosurfaces.

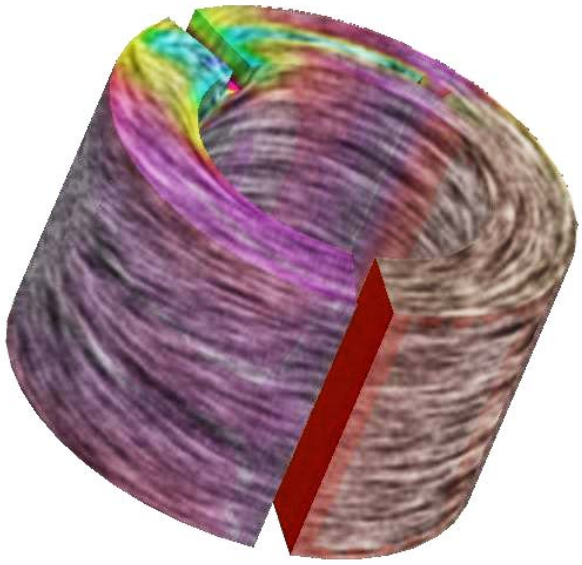


Figure 5.4: The result of applying a higher contrast normal mask to a ring data set. The texture is no longer visible at the inlet of the ring where the texture reflects the flow orthogonal to the surface. Compare with Figure 4.5 on page 52.

We propose an idea inspired by the velocity mask [63], namely, a *normal mask*. A velocity mask can be used to dim or highlight high frequency noise in low velocity regions. Whereas, a normal mask can be used to dim regions of the vector field that have strong cross-flow component to the isosurface. We define the normal mask as:

$$\beta = (\mathbf{v} \cdot \mathbf{n})^m \tag{5.1}$$

where β increases as a function of the product of the velocity, $\mathbf{v}$, and normal vector to the surface, $\mathbf{n}$, at that point. Here, m is an arbitrary number. In practice, it is typically around unity giving the opacity a nearly linear behavior. In our case, the image overlay becomes more opaque in regions with a strong cross-flow component and more transparent in areas of highly tangential velocity. With the normal mask enabled, the viewer's attention is drawn away from areas of strong cross-flow component, and towards areas of high tangential velocity. However, the texture properties are still advected according the velocity vectors projected onto the isosurface.

The result of applying a normal mask to a ring surface is shown in Figure 5.4. The inlet area where the flow is generally orthogonal to the surface has a high opacity covering up the high spatial frequency texture. Note that if we encode the normal mask as opacity, another simulation attribute can be mapped to hue. In our application this is a requirement for consistency.

Normal Mask Implementation

We can integrate the implementation of the normal mask with very little overhead by taking advantage of the graphics hardware. If we look closely at the construction of the velocity image, we note that the the R, G, and B image channels are used to encode the x, y, and z values of the vector components at each vertex defining the surface. This leaves the alpha channel as a free parameter in the velocity image construction. In order to implement the normal mask, we simply store β into the alpha channel of the framebuffer at the same time we are storing the x, y, and z vector components–when rendering the velocity image. And when reading back the image buffer, reading the alpha component in addition to the

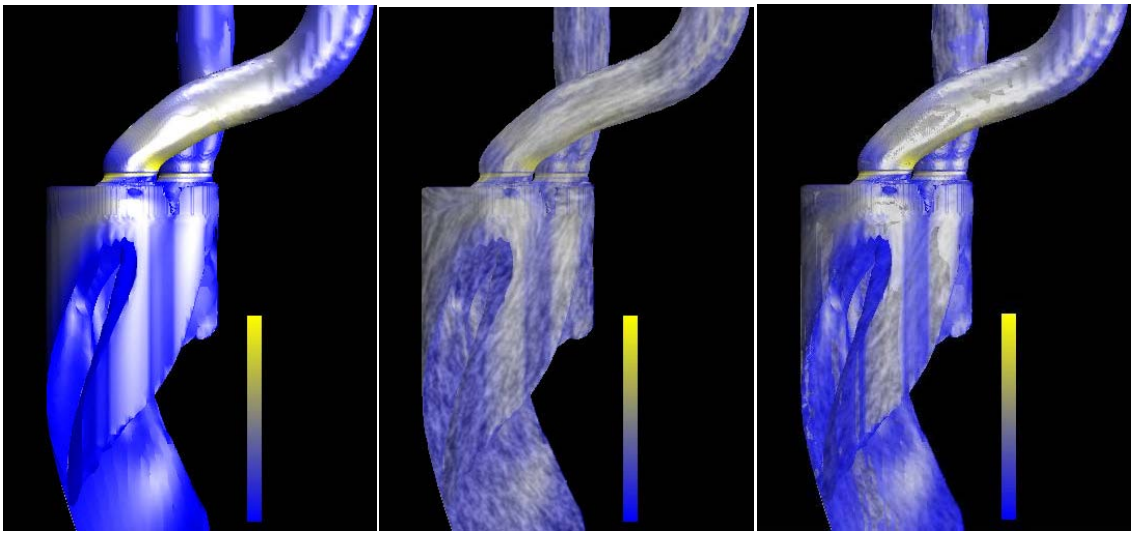


Figure 5.5: (left) a velocity isosurface of value 5.0 m/s with a CFD simulation attribute mapped to hue and (middle) texture-based flow visualization (right) texture-based flow visualization on the isosurface combined with a normal mask. A close-up is shown in Figure 5.8.

R, G, and B component comes at very little overhead.

Some results of applying this normal mask to an isosurface are shown in Figure 5.5. We can see that the flow at the isosurface just below the intake port in the foreground (with a strong specular highlight) has a strong normal component to the isosurface. The higher frequency texture in this region is difficult to see. Figure 5.8 shows a close-up for increased clarity of exposition. Note also that we have chosen a simpler color scale in this case to reduce the visual complexity of the result. We find that using a full range of hue for the color mapping (like in Figure 5.2) in combination with variable opacity for the normal mask is visually complex. So we provide the option of trading off some complexity in the color map while applying the normal mask.

Perceptual Issues

Figure 5.8 shows a close-up view of a velocity isosurface with texture-based flow visualization applied. One perceptual problem with the result is that of occlusion. There is more structure to this isosurface than we can see. Perceptual problems such as occlusion and visual complexity are common to generally all 3D visualizations. One way we addressed this is by implementing an interactive clipping plane. The clipping plane allows the user to see occluded parts of the isosurface by removing sub-sets of the geometry on one side of the plane, in this example, the side closer to the viewer. Again, the users are interested in cutting planes because of their familiarity. Figure 5.6 shows another view of the isosurface with a clipping plane being used. New structures in the isosurface are revealed, namely the structure resulting from flow around an intake port valve.

Of course another alternative is for the engineer to take a 2D slice through the volume, rather than creating an isosurface. This is essentially trading off dimensionality in order to reduce perceptual problems. In our application, the user has both options. Another option for the user in our application is the use of arbitrary clipping geometries. For example, the user can define a clipping geometry in the shape of a sphere or cylinder and apply the texture-based flow visualization. Again however, this is a trade-off. We may gain by lowering visual complexity and occlusion, but we lose some information about the behavior of the flow, namely, that visualized by the isosurface.

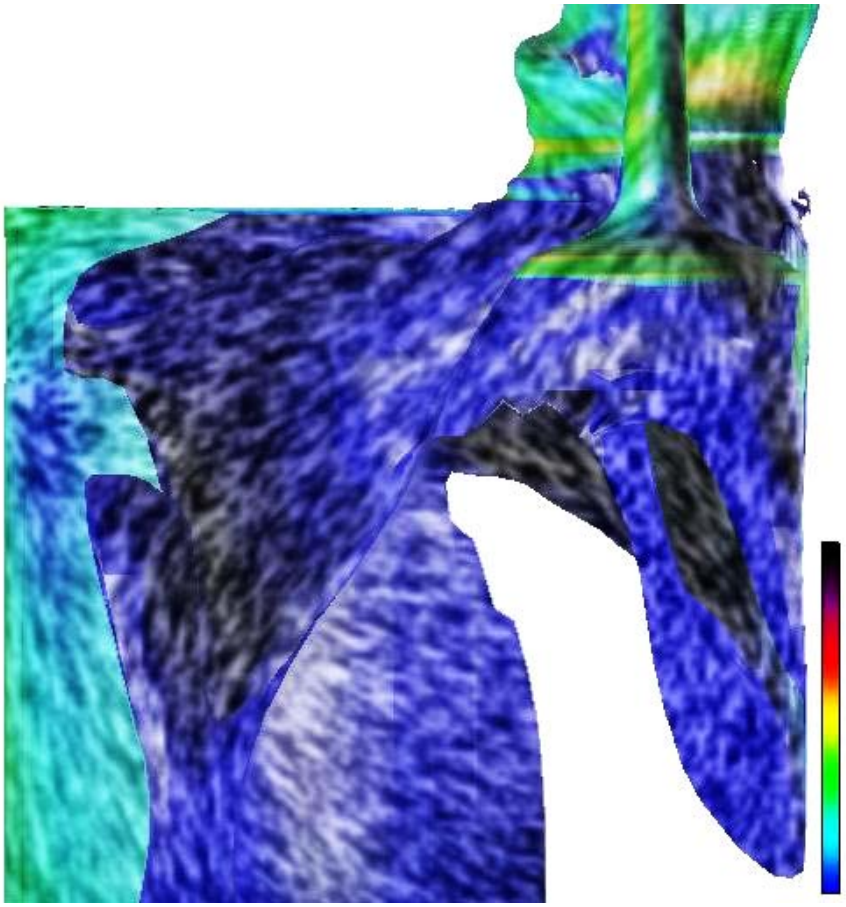


Figure 5.6: A close up view of the same isosurface shown in Figure 5.5 using a clipping plane tangent to the view-point in order to reveal occluded isosurface structures.

A Resampling Point of View

The reason this texture-advection method is faster than previous texture-based methods on surfaces is because the injection, blending, and advection of noise textures is done in image space. The key to transforming the three-dimensional nature of textures on surfaces to a two-dimensional problem is via the projection of the vector field to image space.

The vector field from the isosurface is projected onto image space via the velocity image described in Section 5.3. Then, the image is warped according to a regular, rectilinear mesh. By distorting the image according the projected velocity vectors located at the grid intersections, we are implicitly resampling the vector field. This resampling implies some consequences. Unlike the nature of boundary surfaces, isosurfaces may contain very small, disconnected pieces. In some cases these pieces may only cover a few pixels. This implies that we need a high vector field resampling rate when advecting the textures in image space. In other words, the sampling-to-pixel ratio should not be too small, e.g., sampling at every pixel or every other pixel. In order to handle this, we give the user the option of different advection grid resolutions. In our implementation, the highest grid resolution samples the vector field at every pixel, while the second highest advection grid resolution samples the vector field at every other pixel.

Another reason we give the user control over the resolution of the advection grid is because we want to retain the advantages obtained by decoupling the original surface mesh with the mesh used to advect the

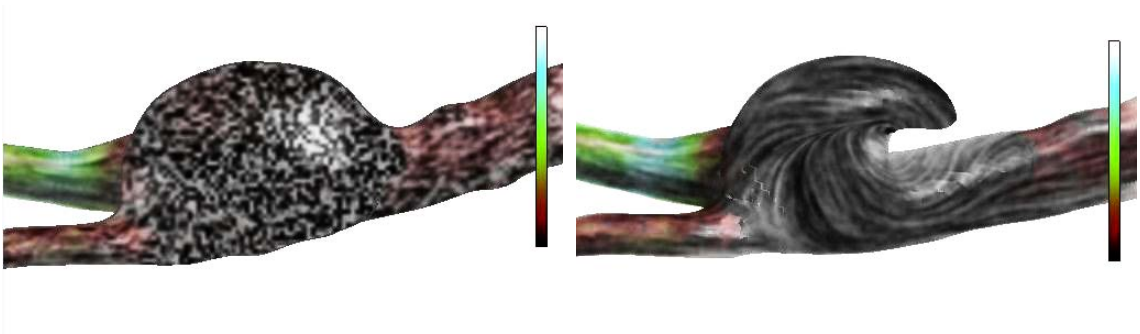


Figure 5.7: (left) the intersection of three blood vessels. An abnormal pocket has formed at the junction. (right) a velocity isosurface of value 0.04 m/s with texture-based flow visualization applied. The recirculation zone where blood flows in the opposing direction becomes clear.

textures. This decoupling prevents computation on those polygons whose area covers less than one pixel. And in the case of Figure 5.3 there are thousands of such polygons. We note also that zooming in on a surface implicitly increases the sampling rate of the vector field because more of the image is spread out while the resolution of the advection grid at the same time remains the same.

The fact that an isosurface may contain many small, disconnected pieces also implies that we need a high frequency texture in the spatial domain. In our implementation, we give the user control over the spatial frequency of the noise injection. Using a high spatial frequency allows for the visualization of flow on even very small, disconnected pieces of an isosurface.

5.5 Results and Discussion

If we take a closer look at Figures 5.5 and 5.6, we can see that the texture-based flow visualization provides additional insight into the behavior of the flow. One of the questions that the engineer poses is: Where in the volume is the ideal swirl flow pattern *not* being met? Within the texture, we can see that the ideal swirl flow pattern is not being met in just below the intake ports themselves. Namely, we can see that two areas of the flow are working *against* each other just beneath the actual boundary surface of the combustion chamber. This is shown more clearly in a close-up in Figure 5.8. The normal mask in Figure 5.8 highlights the boundary between this destructive flow pattern. This is contrasted with only the isosurface itself (Figure 5.2) where area destructive flow is not obvious. The destructive flow pattern is made even more obvious in an animation of the flow [2].

Figure 5.7, shows the intersection of three blood vessels. The larger vessel on the right brings in blood and distributes it to two small vessels on the left. An abnormal pocket, e.g., an aneurysm, has developed at the junction of the three vessels. The observer may be interested to investigate the inside of the pocket to see the resulting blood flow pattern. If we look at the blood flow at the surface, as in Figure 5.7, left, we see mostly noise. The velocity of the blood flow at the surface of the pocket is moving very slow relative to the vessel surfaces. Figure 5.7, right, shows a velocity isosurface (of 0.04 m/s) inside the volume with texture-based flow visualization applied. Shown clearly is the recirculation zone in the pocket with blood flowing upstream (in the opposing direction). This second example was chosen in an effort to support our claim that the hybrid approach of texture-based flow visualization on isosurfaces can be useful, not only in the automotive domain.

[2] Supplementary MPEG animations and images of the results can be found at:
`http://www.vrvis.at/ar3/pr2/VisSym04/`

5.6 Performance

Performance was evaluated on a PC with an Nvidia 980XGL Quadro graphics card, with a 2.8 GHz dual-processor and 1 GB of RAM. The performance times reported in Table 1 support interactive exploration of flow on isosurfaces. This is important for the case of changing isovalues. When the user changes the isovalue, texture updates only require a fraction of a second. And the transition is generally coherent because each frame is blended with the previous frame [83].

num. polygons	advection mesh resolution	FPS
10K	64^2	64 (35)
	128^2	64 (18)
	256^2	32 (8)
	512^2	15 (2.3)
48K	64^2	64 (13)
	128^2	64 (10)
	256^2	32 (6)
	512^2	15 (2)
221K	64^2	64 (4)
	128^2	64 (4)
	256^2	32 (2.9)
	512^2	13 (1.5)

Table 5.1: Sample frame rates for the ISA visualization algorithm applied to isosurfaces.

The first time reported in the FPS column is for the static case, i.e., the absence of changes to the view point. The times shown in parenthesis indicate the dynamic cases of interactive zooming, rotation, and translation of the view point. The reported times are about three times faster than those reported in chapter 4 (and by Laramee et al. [83]). This is mainly due to the updated hardware used for the evaluation and improvements to the implementation.

Normal mask construction does not introduce significant overhead since it is easily built into the advection process itself. For example, the isosurface shown in Figure 5.8 is composed of 243K polygons. In the static case, the normal mask has no effect on frame rates. They are the same as those listed in Table 1. In the dynamic case using a 128^2 advection mesh, the frame rate drops from 3.3 to 3.0 FPS with the addition of the normal mask.

5.7 Discussion and Future Work

We have shown how the image space based texture-advection technique of Laramee et al. [83] can be applied to isosurfaces. Isosurfaces provide information into the 3D characteristics of flow that 2D slices and boundary surfaces alone cannot. We have shown that adding the texture-based representation of flow to isosurfaces can give the engineer new insight into the behaviour of swirl flow when examining CFD simulation data. We have also applied the method to the visualization of blood flow. Furthermore, the method is fast and supports user interaction such as zooming, rotation, and translation.

Future work can go in many directions including visualization of texture-based flow visualization on time-dependent isosurfaces, streamsurfaces, and unsteady 3D flow. Challenges will include both interactive performance time and perceptual issues. Future work also includes the application of more specialized programmable graphics hardware features in the manner of Weiskopf et al. [179]

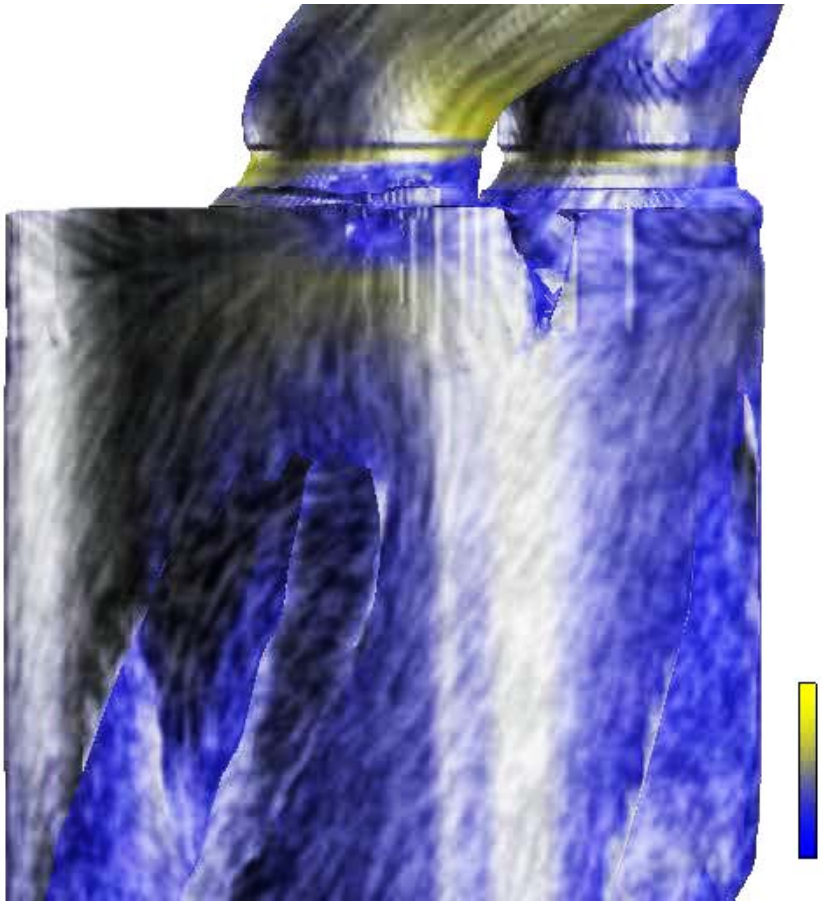


Figure 5.8: A close-up of a velocity isosurface from Figure 5.5: with texture-based flow visualization and a normal mask applied. With the texture advection on the isosurface, it is clear that the ideal swirl flow pattern is not exhibited in this region.

Chapter 6

Geometric Flow Visualization Techniques for CFD Simulation Data

"The strength of the United States is not the gold at Fort Knox or the weapons of mass destruction that we have, but the sum total of the education and the character of our people."

– Pell Claiborne [1] (1918–)

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 measured, experimental results. One pervading message we hear consistently is: users are interested in more interactive control of the flow visualization results–a classic theme in the realm of scientific visualization [57]. 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.

6.1 The Versatility of CFD Grids

Another reason the users request more interaction control over the visualization results is due to the fact that CFD meshes embrace a wide variety of components, features, and levels of resolution. To illustrate, we look at Figure 6.1 showing two intake ports. By looking at an overview, we observe what appear to be four adaptive levels of resolution: (1) for the flow source on the left and the combustion chamber on the lower, right, (2) another level of resolution for the connecting pipes in the middle, and two levels of resolution for the intake port components themselves. When we zoom in (Figure 6.2) we find five adaptive levels of resolution used to evaluate the intake ports themselves: (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 the rings around the base of the ports. The facets in the flow source (on the left in Figure 6.1) are approximately 1000–2000 times larger than the finest resolution facets at the base of the intake

[1] Pell, Claiborne de Borda, U.S. Democratic politician; Senator from Rhode Island 1961–; sponsor & eponym of Pell grants; chairman of Senate Foreign Relations Committee 1987–; great-great-great nephew of George Dallas

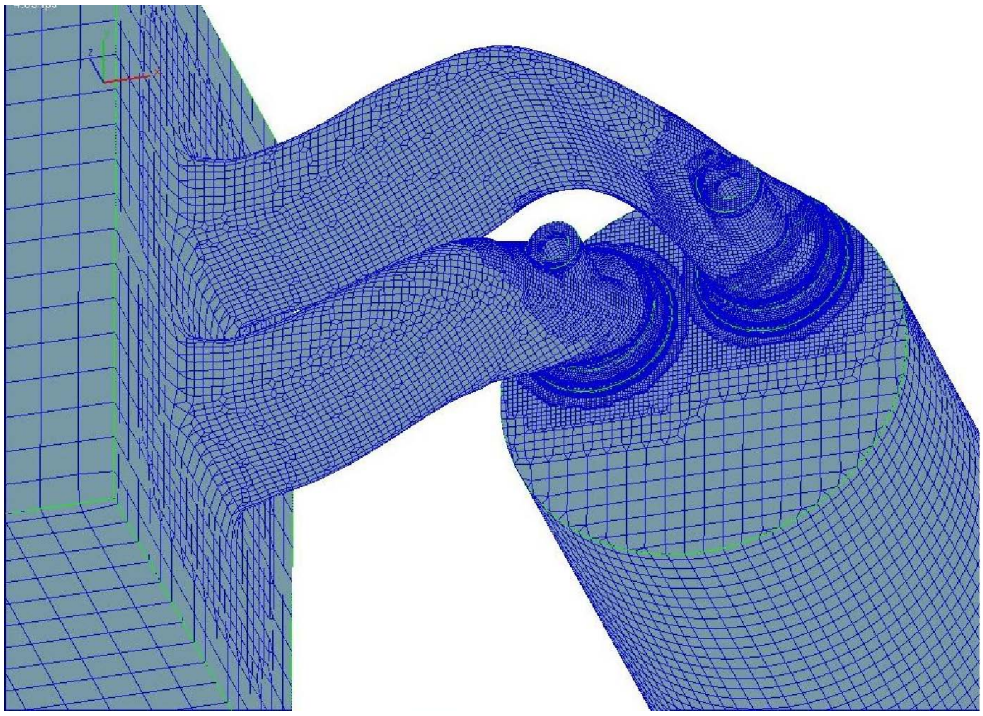


Figure 6.1: The CFD simulation grid of an intake port. This image illustrates the versatility of a typical, unstructured, CFD simulation grid containing a flow source from the left, two connecting pipes in the middle, two intake ports at the ends of the pipes, and a combustion chamber cylinder on the right.

ports. These grids are a daily experience in the industrial CFD community. Our goal is to provide flow visualization solutions that are equally as versatile and adaptive as the grids themselves.

6.2 Perceptual Challenges

A large amount of flow visualization research literature addresses 2D visualization techniques. This is partly because 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. Most of the CFD simulation grids at AVL are unstructured and three dimensional. Although engineers often use 2D slices through the 3D meshes during 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 [171].

The rest of this chapter is organized as follows: In chapter 2 we discussed related research in flow visualization with a section on geometric approaches. Section 6.3 describes our approach of using oriented streamlines and streamlets. Section 6.4 introduces a novel animated streamline technique. Section 6.5 outlines the streamrunner and streamcomet concepts and resulting implementations. Finally, we conclude with some initial results and ideas for future work. We note also that portions of this chapter have been published previously [77, 80].

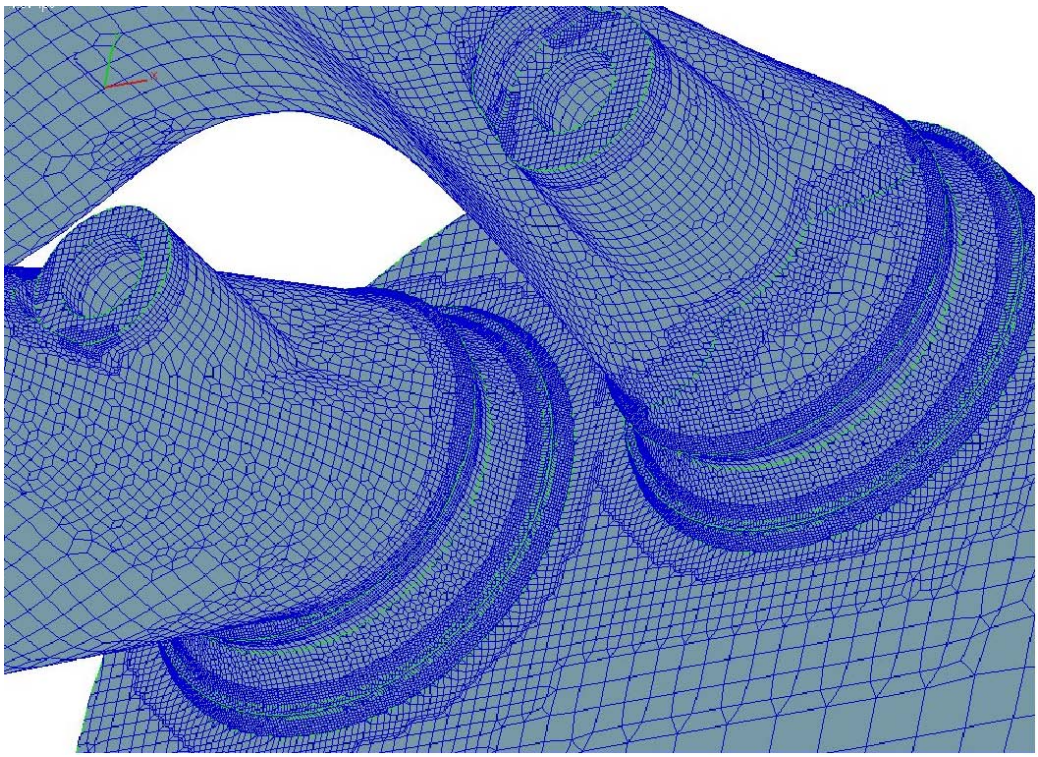


Figure 6.2: A close-up view of the intake ports in the same CFD simulation grid as shown in figure 6.1. The mesh contains multiple, adaptive resolution levels of unstructured grid cells.

6.3 Oriented Streamlines and Streamlets

One of the drawbacks of conventional streamlines is the lack of flow orientation (upstream vs. downstream direction) depicted in a still image. Our system incorporates an oriented streamline implementation. Oriented streamlines convey the downstream direction of the flow by varying the opacity as a function of particle trace evolution. In other words, the further downstream an integration path is traced, the higher the opacity of the streamline. This can be implemented by giving the streamlines a finite width, either automatically or through user-defined parameters, and using semi-transparent polygons in order to depict an oriented streamline (Figure 6.3, middle). Arrow heads could also be used to achieve the same effect. However, arrow head glyphs can lead to visual clutter without careful treatment.

Careful attention must be paid when rendering oriented streamlines on boundary surfaces in order to prevent artifacts resulting from overlapping streamline and boundary surface polygons. These artifacts can be avoided through the use of OpenGL's polygon offset functionality. The result is similar to that of OLIC (Oriented Line Integral Convolution) [176, 175]. One important difference is that OLIC is based on a traditionally slower approach derived from LIC. Also, OLIC is more suitable for the visualization of 2D vector fields.

For the case of unsteady flow, drawing a continuous particle path using only a single time-step from the data set can be considered misleading. This is because no particle actually traces such a path. For the case of slices and surfaces, the visualization becomes even more problematic because a component of the vector field is taken away, namely that component orthogonal to the slice or surface, absent after a projection onto the slice or surface. One approach to handling this is through the use of streamlets (short streamlines). Figure 6.3, left-to-right, shows the use of streamlines, oriented streamlines, and streamlets

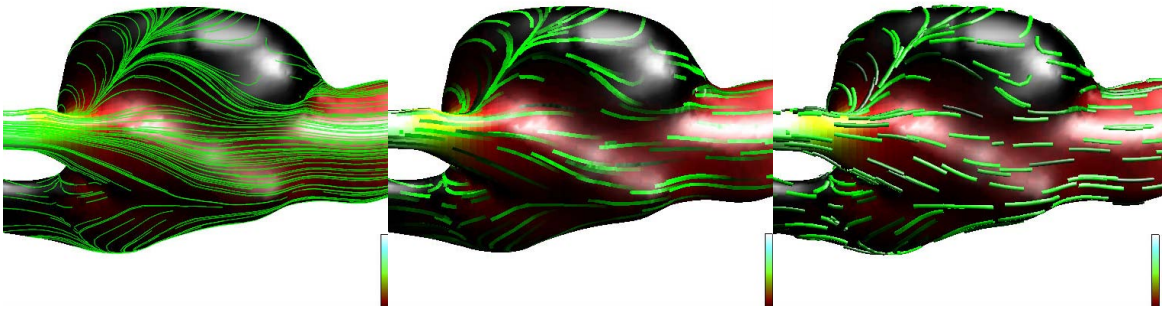


Figure 6.3: The visualization of blood flow at the surface of an aneurysm: (left) geometric flow visualization using streamlines (middle) oriented streamlines and (right) streamlets.

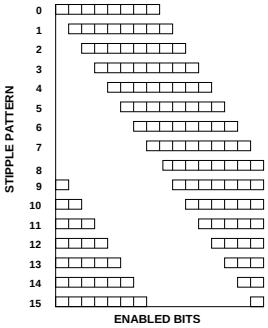


Figure 6.4: The 16-bit stipple pattern series used for animating streamlines in real-time, based on OpenGL 1.1.

all applied to the same data set. The data set in this case is simulation data coming from blood flow through an aneurysm.

6.4 Animated Streamlines

Here, we use a stippling approach to animate streamlines such that the downstream direction of the flow is depicted. The advantage here is that the stippling approach is supported by OpenGL 1.2. and commodity graphics hardware. Thus real-time frame rates can be achieved even for large numbers of streamlines. Anti-aliasing, also supported by the graphics hardware, can be added to visually enhance the results at very little overhead.

We apply a line stipple pattern to streamline paths. Each streamline is rendered using one of 16 stipple patterns shown in Figure 6.4. In order to add animation, we simply shift the stipple pattern applied to the integral paths at rendering time [2]. This approach is reminiscent of that used by Jobard and Lefer [66] or Berger and Gröller [7] where a color-table look-up approach is used to animate the streamlines. One important difference is that the technique here applies well to 3D flow.

Without special handling, geometric techniques can also suffer from some of the same perceptual problems that direct flow visualization can. One means by which to focus on a particular subset, area of

[2] For supplementary images and MPEG animations, please visit:
`http://www.VRVis.at/ar3/pr2/geometricApproach/`

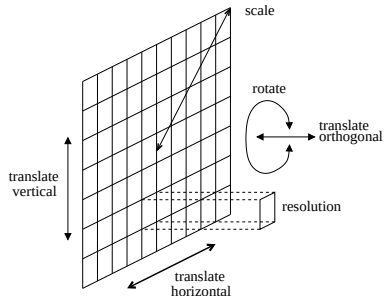


Figure 6.5: Our seeding plane implementation has six interactive DoFs: (1-3) three translational, (4) scaling, (5) rotation, (6) resolution.

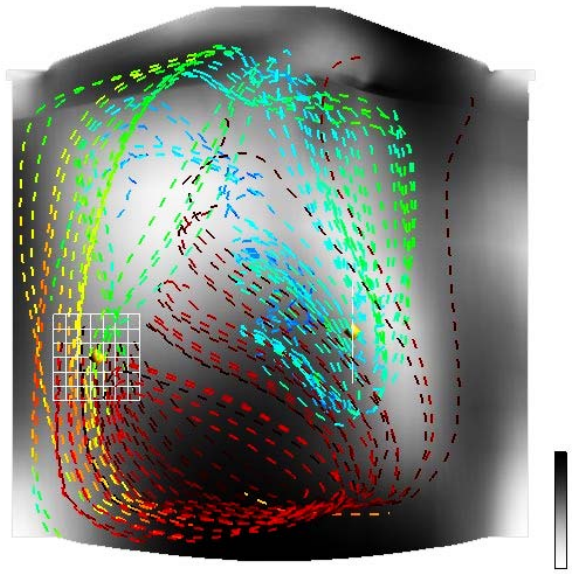


Figure 6.6: The visualization of tumble motion using animated, dashed streamlines. Two seeding planes are used: one seeding color mapped streamlines, the other emanating red streamlines. A black and white-mapped slice serves as context information.

interest, or feature of a flow field is via a streamline seeding strategy. In general, three popular streamline seeding strategies are often used: (1) *image-based* seeding strategies such as that described by Turk and Banks [159] or the evenly spaced-streamline seeding strategy presented by Jobard and Lefer [65], (2) *topological* or feature-based, seeding strategies such as those presented by and Löffelman and Gröller [91], Sanna et al. [128], or Verma et al. [169] and (3) *interactive* seeding strategies using a streamline seeding rake used by Bryson and Levit [14] or Schultz et al. [137]. Our approach falls into the third category–an interactive streamline seeding strategy. Users would like full control over which subsets of the vector field to highlight in order to highlight both desirable and undesirable characteristics of the flow.

A schematic of our interactive streamline seeding tool is shown in Figure 6.5. This tool provides the user with six interactive degrees of freedom (DoF): (1-3) three translational, (4) scaling, (5) rotational, and (6) resolution control. These interactive DoFs are required to investigate the results of CFD simulations because the meshes from CFD embrace a wide variety of components, features, and levels of resolution. Ideally, the tools used to analyze and visualize these data sets should be flexible enough to adapt their

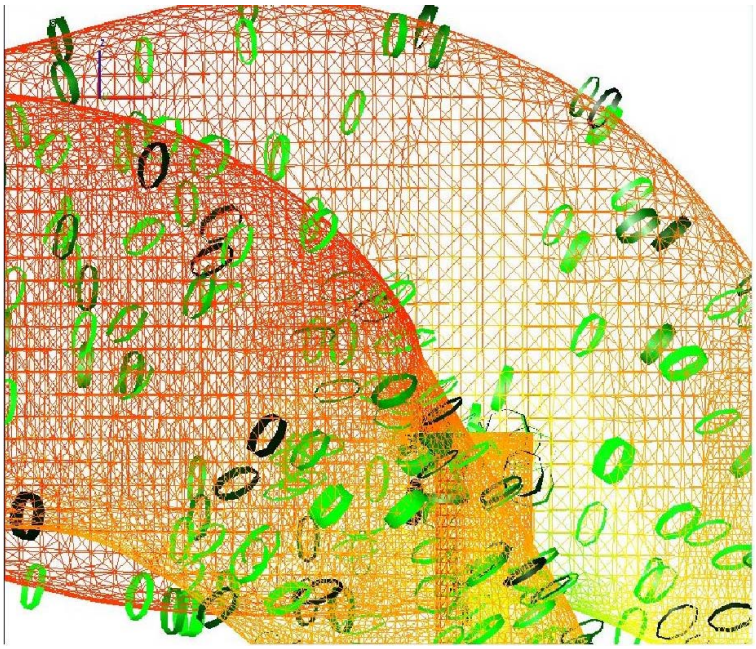


Figure 6.7: This image shows stream seeds as short pipe segments including a wire-frame context of the connecting pipes in the intake ports data set. In this way occlusion and image complexity are minimized.

size, orientation, and resolution to fit the features of interest either automatically or through user-specified parameters.

Figure 6.6 shows animated-dashed streamlines used to visualize tumble motion [86]. The sparser animated-dashed streamlines allow the user to see through the volume. Furthermore, the implementation is simpler than the dash tube technique of Fuhrmann and Gröller [47].

6.5 Streamcomets

Streamcomets are an extension of the streamrunner [77]. The streamrunner addresses the problems of occlusion and scene complexity by giving the user control over the evolution of streamlines from seeding time until they terminate. A streamline may terminate when it reaches a boundary in the geometry, reaches a region of zero velocity, or reaches a maximum length set by the user. The two interactive DoFs afforded by the streamrunner are: (1) the position of the stream's head along the integral path and (2) the diameter of the the integral object, in this case the tube diameter.

Using the streamrunner, the user is able to set the stream evolution to its origin as shown in Figure 6.7. In this figure, only the seeds are shown. Individual streamlines are easily distinguished and focused upon early in their evolution because occlusion has been almost eliminated while visual complexity is at a minimum. The streamrunner can then be used to change the current geometric length of the shaded tubes such that the user can watch the streamlines grow, or *run*, in the direction of the flow. This gives a clear, unequivocal indication of flow direction. The user is able to focus on an individual streamline, a group of streamlines, or a particular area of the flow as users adjust the current geometric length. Watching the streams flow in 3D combined with shading, gives added depth cues. The streamrunner also allows the user to trace the evolution of the streamlines *backwards* in order to see where a path originated.

Streamcomets follow a very intuitive metaphor. They offer four interactive DoFs as shown in Figure

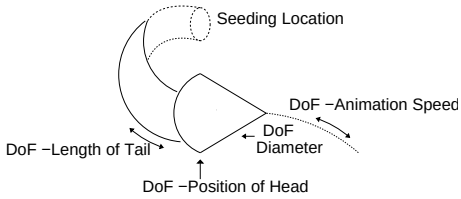


Figure 6.8: The streamcomet promotes 4 interactive degrees of freedom: (1) the position of the comet head along the path of integration, (2) the diameter of the comet head and tail, (3) the length of the comet tail, and optionally (4) the animation speed of the comets

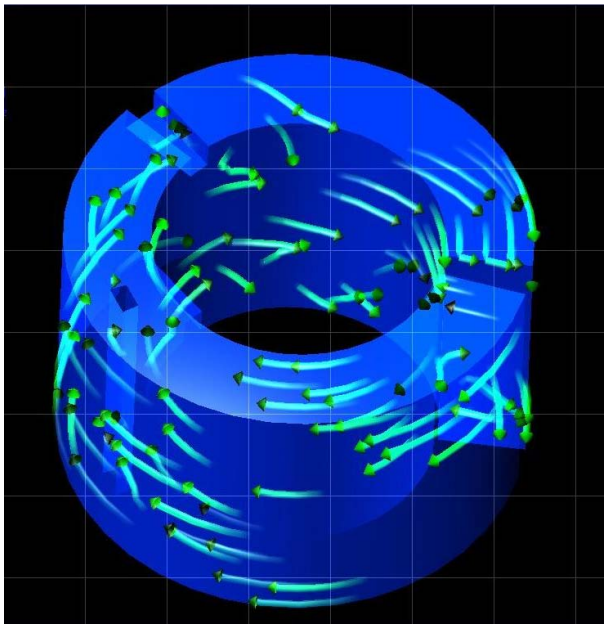


Figure 6.9: Here streamcomets are rendered in the context of a semi-transparent ring geometry. We add semi-transparency and a glowing impression to the streamcomet tails, whose transparency increases with the distance from the comet head.

6.8. The user is given interactive control over: (1) the position of the head along the integral path, (2) the diameter of the comet head and comet tail, (3) the length of the semi-transparent comet tail, and optionally (4) the animation speed of the comet along the path of integration. Coupled with more interactive degrees of freedom, streamcomets offer the advantage of showing local flow direction and curvature for static images. There is strong evidence to support the notion that flow visualization objects that show the direction of the local vector field improve the user's ability to identify critical points and understand particle advection paths [75].

Figure 6.9 gives us an impression of what it is like to use the streamcomets for 3D flow visualization. We include the semi-transparent ring geometry as context information. We also apply a semi-transparent function to the comet tails and give them a glowing effect. The alpha value along each comet tail is a function of the distance to the comet head i.e., the further away from the head, the more transparent the tail.

Another useful feature is the option of animating the streamcomets. Conceptually, animating the streamcomets such that the comet head position is automatically incremented along the path of integration, acts as a visual search function. The viewer is able to use the animation to search for optimal comet head

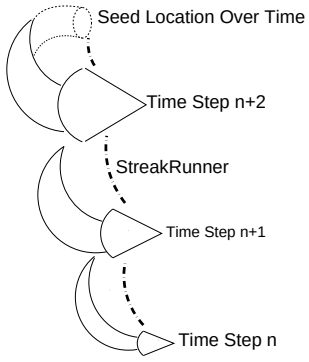


Figure 6.10: The use of the streamrunner and streamcomets for unsteady flow visualization. Comet heads shrink over time. i.e., the older the comet, the smaller the comet head. The interactive equivalent of a streakline is a streakrunner, which interactively controls the number of discrete time steps along the streakline defined by the series of comet heads.

positions. This is very useful when the user is:

- not sure where to position the head,
- searching for interesting features in the flow field, or
- optimizing the other interactive degrees of freedom.
- We also give the user the option of interactively adjusting the animation speed.

We do not propose the streamrunner and streamcomet as stand-alone features. They are meant to be combined with other classic, 3D interaction techniques such as rotation, scaling, and translation. Additional important features we have included are: the option of choosing a non-uniform coloring scheme so colliding geometric objects can be more easily distinguished, turning on or off semi-transparent or wire-frame context information, and adjusting the streamline seeding density in the flow field.

The comet glyph can intuitively encode time attributes for unsteady flow visualization. At the top of Figure 6.10, we see a sample seed point whose geometric location is constant over time and from which streamcomets are injected into the flow, similar to a *streakline* – the line traced by a set of particles that have previously passed through a unique point in the domain [136]. As the comet ages (after being injected into the flow), the size of the head decreases as does a real comet when traveling through space. Also the length of the tail encodes the local instantaneous velocity at the comet's current position. The color of the comet head encodes the local temperature and the color of the comet tail reflects another scalar attribute of the flow such as pressure. If we represent comet tails using streamtubes [160],, the local convergence and divergence of the flow may be encoded. If comet tails are represented using streamribbons [160], local vorticity is encoded. Ideally, the user is able to toggle between the two representations. We claim that the use of streamcomet glyphs for encoding attributes of the flow is more intuitive than using other glyphs such as superquadric shapes [38]. The interactive analogue of a streakline is a *streakrunner*. The streakrunner is an interactive control that defines the geometric length of the streakline. Such a line is shown in Figure 6.10 connecting the comet heads.

6.6 Results

Performance times depend on the number of streamlines. Performance times for the animated streamlines are given in Table 6.1. Performance was evaluated on a machine running Red Hat Linux 7.2 with a 1 GHz Pentium III dual processor, 1 GB of RAM, and an *HP fx* graphics card. Note that the frame rate also varies as a result of caching. Anti-aliasing adds very little overhead since it is built into OpenGL 1.2 and hence

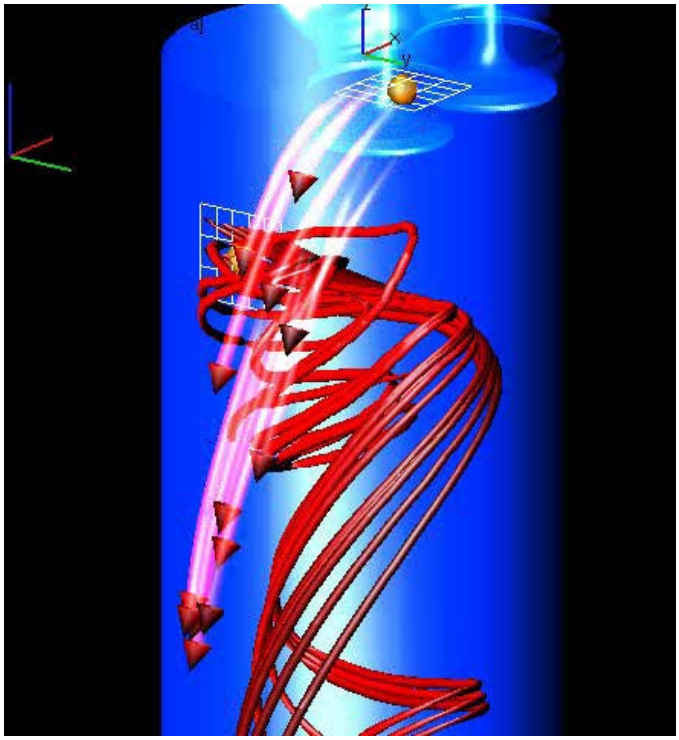


Figure 6.11: Two seeding planes in the combustion chamber of a piston valve: one seeding streamcomets, the other seeding shaded streamlines.

is supported by most graphics cards. As we see, the stippling approach allows animation of thousands of streamlines in real-time. Furthermore, we have not employed the more modern graphics cards or display lists to increase the frame rates. Figure 6.11 shows two seeding planes inside the combustion chamber

no. of streamlines	with anti-aliasing	without anti-aliasing
10	70	70
100	54	60
1,000	25	27
2,500	15	18
5,000	6	7
10,000	4	5

Table 6.1: Sample frame rates for the animated streamlines (in frames per second).

of a piston valve. The seeding plane in the top (foreground) has streamcomets emanating from it. The seeding plane in the middle (background) seeds ordinary streamlines. We emphasize the importance of the user's ability to resize the streamcomets along arbitrary dimensions when zooming in and out of the data sets. It is important to note that changes to the diameter of the comet heads apply to the entire collection of streamcomets, and are not applied on a per-comet basis. Applying size changes to individual comets would lead to misleading visualization results, e.g., the user may interpret different comet head sizes to be a reflection of scalar properties inherent in the flow field.

Figure 6.12 gives us another impression of what it is like to use the streamcomets for flow visualization in 3D. Here, both streamlines and streamcomets are used to visualize a vortex. Giving the user interactive

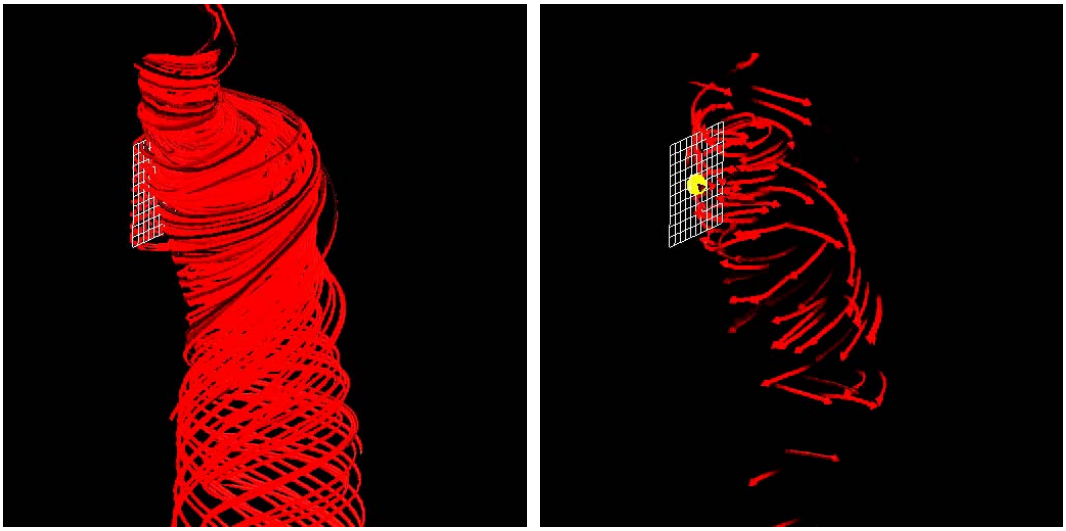


Figure 6.12: The visualization of a vortex using (top) streamlines and (bottom) animated streamcomets. The streamcomets reduce occlusion and provide the same coverage when animated.

control over the placement of the comet heads, the diameter of the comet heads and tails, the seeding density, and the length of semi-transparent comet tails, affords the user a very good opportunity to see the characteristics of the flow field.

6.7 Discussion and Future Work

The added interaction provided by our geometric flow visualization techniques is very useful for flow visualization in 3D and within the domain of versatile grids associated with CFD simulations. This is because they are based on geometric primitives that are more suitable for the visualization of 3D flow than approaches based on color-mapping, glyphs, or textures only. The user control afforded by the streamcomets as well as the intuitive metaphor on which they are based makes them more versatile for 3D flow visualization than previous techniques. Furthermore, the simplicity of our approaches makes them strong candidates for inclusion in other flow visualization software packages. The approaches described here have been included in a cross-platform, industry-level visualization application for the analysis of CFD simulation data. These geometric objects give a new level of control over to users investigating a vector field. We encourage the reader to view the animations at the given URL. [3]

Future work could go in several directions including: (1) an implementation prototype of the streamrunner and the streamcomet for unsteady flow visualization including the introduction of a *pathrunner* – the unsteady equivalent of a streamrunner, a streakrunner – the interactive equivalent of a streakline or (2) a formal HCI evaluation of the perceptual effectiveness of the streamrunner and streamcomets for 3D flow visualization – an extension of the work by Laidlaw et al. [75] to include more flow visualization techniques.

[3] `http://www.VRVis.at/ar3/pr2/geometricApproach/`

Chapter 7

Investigating Swirl and Tumble Flow with a Comparison of Visualization Techniques

"The only stable state is the one in which all men are equal before the law."

–Aristotle [1] (384 BC–322 BC)

Previously, AVL engineers used a series of several color-mapped slices to assess and visualize the results of their CFD simulations. Recently, new solutions for the visualization of CFD simulation data have been introduced in order to overcome some of the limitations outlined in Chapter 2. We report on the application of these techniques in addition to the more traditional approaches. In this chapter we describe: (a) the application of different visualization techniques to specific application cases, (b) advantages and disadvantages of what these techniques offer, and (c) a comparison which may apply to other application cases. We also to give recommendations on when to use specific techniques and in which application scenario.

7.1 Evaluating Swirl and Tumble Motion

For many of the automotive components that undergo evaluation, there is an ideal pattern of flow the engineers are trying to create. In the flow within a cylinder, we can distinguish between two types of motion: swirl flow commonly found in diesel engines and tumble flow commonly found in gas engines. In both cases, rotational motion occurs about an axis, though the position of the respective axis is different. In the case of swirl flow, the axis is more or less coincident with the cylinder axis, as shown in Figure 7.1, left. In the case of tumble (Figure 7.1, right), the rotation axis is perpendicular to the cylinder axis and more complex, thus making tumble flow more difficult to control than swirl flow. We refer to the rotational axis associated with tumble motion as the z axis that, in this case, points out of the paper towards the reader.

In order to generate swirl or tumble motion, fluid enters the combustion chamber from the intake ports. Later on in the engine cycle, the kinetic energy associated with this motion is used to generate turbulence for mixing of fresh oxygen with evaporated fuel. The more turbulence generated, the better the mixture of air and fuel, and thus the more stable the combustion itself. By stable we mean achieving the same conditions for each engine cycle. Ideally, enough turbulent mixing is generated such that 100% of the fuel is burned. The swirl or tumble motion should be maximized to maximize turbulence. From the point

[1] Greek critic, logician, naturalist, philosopher, physicist, and zoologist, father of logic, father of dramatic criticism, founder of Lyceum, wrote "Physics", "Nicomachean Ethics", "Politics", "Poetics"

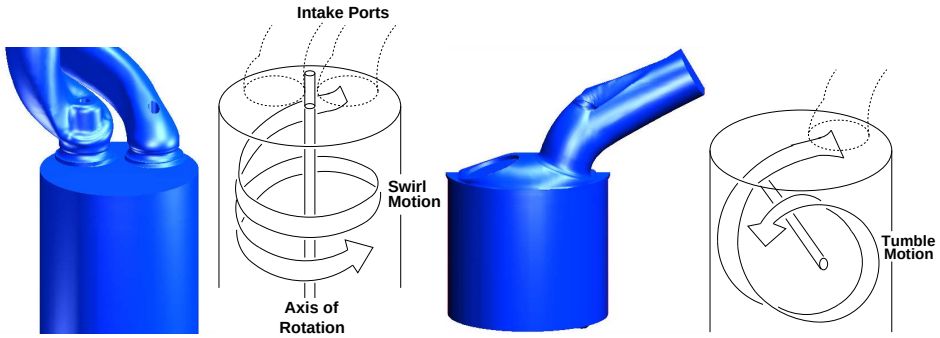


Figure 7.1: (left) The swirling motion of flow in the combustion chamber of a diesel engine. *Swirl* is used to describe circulation about the cylinder axis. The intake ports at the top provide the tangential component of the flow necessary for swirl. The data set consists of 776,000 unstructured, adaptive resolution grid cells. (right) Some gas engine components require a *tumble* motion flow pattern (right) in order to mix fluid with oxygen. Tumble flow circulates around an axis perpendicular to the cylinder axis, orthogonal to swirl flow. The data set is composed of 61,700 unstructured, adaptive resolution cells.

of view of the mechanical engineers designing the intake ports, the ideal flow pattern leads to beneficial conditions including: improved mixture preparation, a higher EGR (Exhaust Gas Ratio) which means a decrease in fuel consumption, and lower emissions. However, too much swirl (or tumble) can displace the flame used to ignite the fuel, cause irregular flame propagation, or result in less fuel combustion.

As such, a balance must be achieved between generating enough swirl or tumble flow and not displacing the flame used to ignite the flow. A controlled flow motion is used to get stable and reproducible conditions at each engine cycle.

Investigating Flow Patterns with Visualization

Central to this study are some routine questions that engineers may ask when investigating swirl and tumble flow:

1. Can visualization provide insight into or verify the characteristic shape and behavior of the flow?
2. What tools can help to visualize the swirl and tumble flow patterns?
3. Where in the combustion chamber are the ideal swirl and tumble flow pattern *not* being realized?

Here we seek answers to some of these fundamental questions using direct, geometric, and texture-based flow visualization techniques. We proceed with a visual analysis of the simulation data from AVL's CFD solver. In particular, we analyze the simulation data from two important in-cylinder flow motion patterns, typical of a mechanical engineer's analysis. Our investigation also carries forth across multiple spatial dimensions, namely in 2D, 2.5D, and 3D. By 2.5D we mean surfaces through 3D space. We note that the large size of the data set shown in Figure 5.1, 776,000 unstructured, adaptive resolution cells, makes unsteady flow visualization very difficult with our hardware.

Direct, Geometric, and Texture-Based Flow Visualization

Our flow visualization classification is described in Section 1.2. In this chapter we use advection approaches according to Image Based Flow Visualization (IBFV) [164] and Image Space Advection (ISA) [83], which can generate both Spot Noise [163] and LIC-like [17] imagery. These approaches are related to Lagrangian-Eulerian Advection (LEA) [64]. We note that a full comparison of texture-based flow visualization techniques [82] is given in Chapter 2.

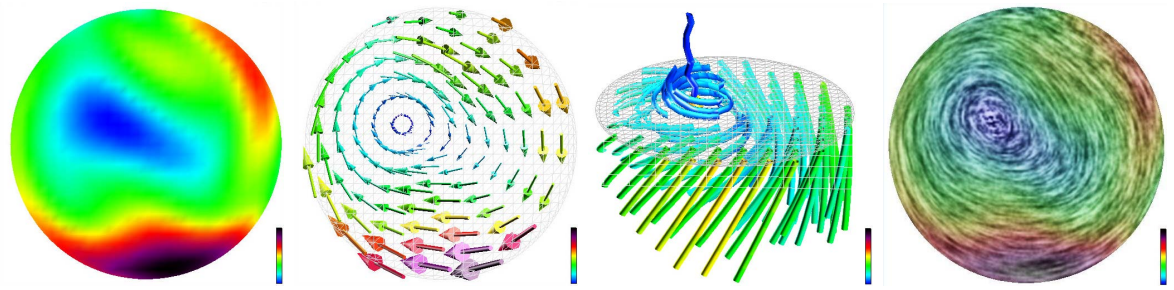


Figure 7.2: The visualization of swirl flow on a slice through the combustion chamber of a diesel engine: (left) direct visualization using color mapping, (middle-left) direct visualization using glyphs, (middle-right) geometric flow visualization using streamlets, and (right) texture-based flow visualization.

We focus only on interactive visualization techniques because an interactive exploration of parameter space is essential for improving the design of intake ports and valve cylinders (e.g., changing isovalues or placing seed points of particles). Hence *feature-based flow visualization*, another class of techniques including feature extraction and tracking is not a focus of ours. Another problem is that each engine component has its own characteristic flow motion. Implementing a feature extraction algorithm for each flow pattern is simply not practical. Further complicating the matter is that the ideal flow motion is only approximated. Post et al. [117] cover feature-based flow visualization in detail.

The rest of the chapter is organized as follows: Sections 7.2, 7.3, and 7.4 investigate properties of the flow patterns using 2D slices, surfaces (referred to as 2.5D), and 3D and hybrid techniques respectively. Each spatial domain is analyzed using direct, geometric, and texture-based visualization tools. These approaches are then compared to one another. Section 7.5 presents a discussion and offers some overall perspectives with respect to this investigation and Section 7.6 outlines our conclusions. We note that the material in this chapter has also been published elsewhere [86].

7.2 Visualizing Flow Motion on 2D Slices

Slices are a common tool used to investigate the properties of the flow inside a volume. One reoccurring theme is that of placement: We must decide where to slice through the volume. This decision is influenced by two factors: (1) our knowledge and experience regarding the data set and (2) some trial and error. The average user in this case is a mechanical engineer with a strong background and previous training in the area of CFD. These users generally have *a priori* knowledge about the data they are investigating. In other words, they already have an idea and associated expectations of the behavior they would like to see. Nevertheless, after an initial slice is generated, further refinement of the visualization parameters might be needed for the desired results.

Direct Flow Visualization

Figure 7.2 shows a 2D slice through the cylindrical combustion chamber from Figure 7.1 left, within a plane perpendicular to the axis of rotation. In this particular case, each approach indicates that the flow has desirable properties, namely, a rotational pattern about the axis parallel to the cylinder, consistent with swirl motion.

Figure 7.2 left shows velocity magnitude mapped to hue. The colors help to differentiate regions of the flow and give an overview of the data. However, the path of the flow is not conveyed clearly. Figure 7.2 middle-left shows shows the same data visualized with 3D glyphs placed on a 2D slice. In this case, the data has been resampled onto a regular grid with a user-specified resolution. A brute-force hedgehog

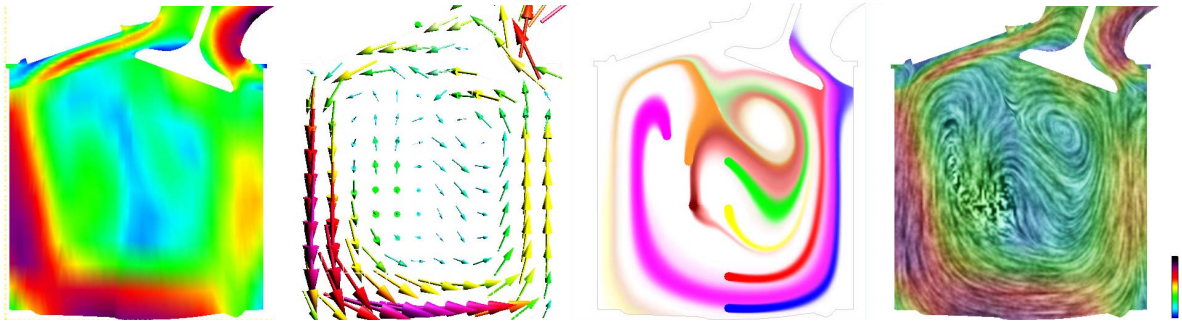


Figure 7.3: The visualization of tumble flow on a slice: (left) direct visualization using color mapping, (middle-left) direct visualization using glyphs, (middle-right) flow visualization using dye injection, and (right) texture-based flow visualization.

technique results in visual clutter and over complexity. We have used the resampling tool to create a polar grid and interactively position the center of the grid such that it aligns with the vortex associated with the swirl flow. The glyphs give an overview of the vector field, show the direction of the flow, and also afford the visualization of the normal component of the flow with respect to the slice, an ability important for applications from the automotive industry [132]. This third component is missing from two of the other visualizations. However, what the glyphs lack is spatial continuity. There is always a trade-off between the resolution of resampling grid and coverage of the flow. Higher resolution resampling decreases the likelihood of overlooking a feature of the vector field, but increases perceptual problems.

Figure 7.3 middle-left, shows direct visualization techniques applied to a slice in order to visualize tumble flow. The glyphs suggest that the flow pattern on the slice is generally consistent with tumble flow. We see an overall circular, counter-clockwise flow about an axis pointing out of the plane. However, some of the features of the flow are higher in spatial frequency that the glyph placement provided by the resampler. If we increase the resampling resolution, the result is difficult to interpret because the glyphs collide in visual space. Color coding, as in Figure 7.3 left, is much less suitable for tumble flow than for swirl flow because the structure of the flow is more complex and cannot be visualized by the magnitude of the velocity.

Geometric Flow Visualization

In general, geometric techniques like streamlines provide more spatial continuity than glyphs or texture-based flow visualization. Texture-based methods require a long convolution filter length to achieve similar spatial continuity. For the case of swirl flow, we find 2D streamlines restricted to the slice misleading because there is a strong flow component orthogonal to the slice, i.e., these long streamlines have less correspondence to physical motion. Hence, Figure 7.2 middle-right shows the vector field visualized with 3D streamlets, namely, short streamlines. These streamlets show the strong component of the flow orthogonal to the slice and introduce more spatial continuity than the glyphs. However, while streamlets do increase spatial continuity they share the same disadvantage as the glyphs in terms of seeding and perceptual challenges.

Figure 7.3 middle-right shows tumble flow on slice visualized using dye injection. The result resembles streaklines which are the same as streamlines for the case of steady-state flow. This particular implementation is done via dye injection in a manner similar to IBFV. The result has some characteristics from both the geometric and texture-based classes of techniques. The borderlines between dye color and white are geometric in the sense that they meet the definition of a streamline: a geometry that is everywhere tangent to the flow. On the other hand, the implementation is realized using textures which introduces

diffusion into the result. Hence this technique could also be classified as texture-based.

The advantages here are that these objects show the downstream and upstream direction of the flow especially clear in an animation and the user may interactively select the seed points and dye color. Hence, the user may use these dye sources to highlight certain areas of the flow such as the vortex associated with the swirl flow. In this case, the flow component normal to the slice is not as strong compared with Figure 7.2. The disadvantage here, and with geometric techniques in general is that placement is crucial. Important features maybe overlooked depending on the spatial position of the objects. Also we must use caution when interpreting these results because no particle actually traverses a path shown in Figure 7.3. The original CFD simulation data is 3D and time-dependent. We note that in this study, we focus on steady-state flow. The data set used to study swirl motion is too large for our old hardware.

Texture-Based Flow Visualization

Figure 7.2 right shows the swirl motion with texture-based flow visualization applied. The advantages of this approach include: (1) complete coverage of the flow field, (2) spatial continuity, (3) velocity magnitude may be encoded in the texture, leaving hue available to include another CFD simulation attribute such as temperature, and (4) flow orientation (upstream and downstream direction) is clear in an animated sequence. In general, computation time is a disadvantage with many texture-based flow visualization techniques. Recently, this hurdle has been overcome with some techniques [83, 84, 164, 178]. However, this approach does not generally provide as much spatial continuity as geometric techniques.

In Figure 7.3 both the dye injection and the texture-based flow visualization techniques reveal that the vector field has properties that deviate from the ideal tumble flow pattern. These techniques indicate the presence of a saddle point in the flow near the top, center of the geometry (see Helman and Hesselink [54] or Post et al. [117] for a more thorough description of flow topology). The deviance from the ideal tumble motion is a result of a trade-off made in the design of the intake port, as its shape must also allow enough incoming fresh air to mix with fuel. Thus air is let in from both sides of the intake port (left and right in this slice). The ideal tumble flow corresponds more closely to one central vortex in the flow. The orange and green dye sources can help us to highlight the saddle point while the texture-based approach ensures that the feature is not overlooked by providing a result with a dense representation across the complete slice.

We also note that the geometric and texture-based visualizations highlight the important asymmetric nature of the flow. There is a counter-clockwise looping pattern near the boundary of the geometry. This is especially clear in an animation. If the flow were too symmetric, the looping pattern might disappear leading to too much destructive flow motion.

7.3 Swirl and Tumble Flow Visualized on Surfaces

Examining the properties of the flow on boundary surfaces (referred to also as 2.5D) is useful because we sometimes start our investigation at the surface in order to get an overview of the vector field before looking inside the volume.

Direct Flow Visualization

Figure 7.4 shows the flow at the surface from the diesel engine component from Figure 7.1, left. In this case, color mapping alone already reveals that the flow has swirl motion characteristics at this instant in time. We see regions of color forming a loose spiral pattern consistent with Figure 7.1, left.

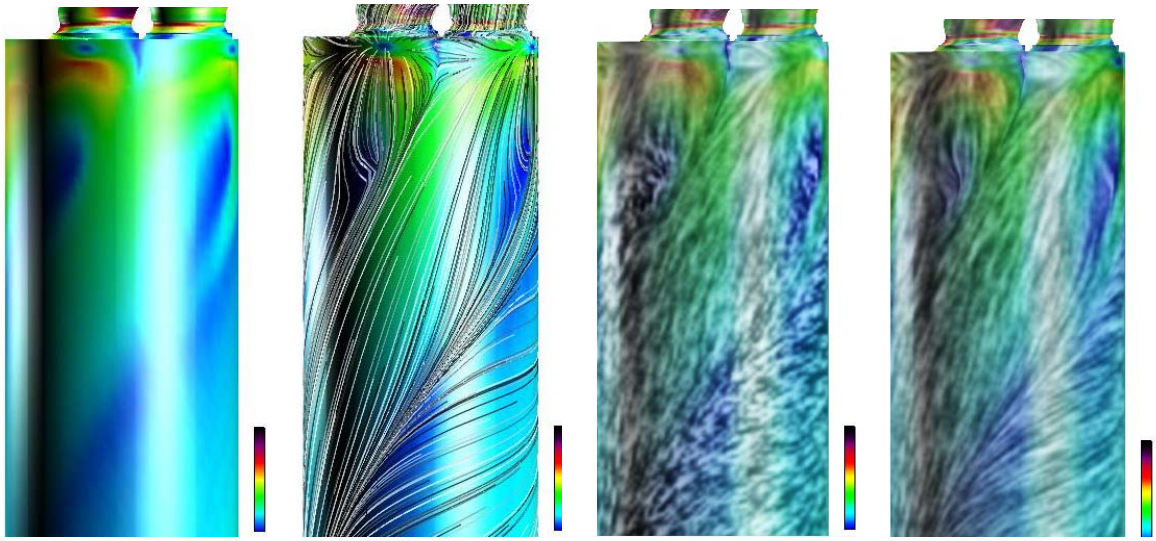


Figure 7.4: The visualization of swirl flow at the boundary surface of the combustion chamber: (left) direct visualization using color mapping, (middle-left) geometric flow visualization using streamlines, (middle-right) spot noise-like texture-based flow visualization, and (right) LIC-like texture-based flow visualization in combination with a velocity clamp.

Figure 7.5 shows the visualization of tumble flow at the surface. From color-mapping alone, it is difficult to answer our questions from Section 7.1. The pattern of flow is more complicated than in the case of swirl. However, color-mapping is still useful in order to identify extremal properties of the simulation data, such as very low velocity magnitude, characteristics that we are generally very interested in avoiding.

Geometric Flow Visualization

Figure 7.4 middle-left illustrates the use of streamlines in order to visualize the properties at the surface. In this case, the implementation computes explicit integral paths according to a particle tracing algorithm. We have added a gray scale phase shift to the hue of each streamline to distinguish individual streamlines that collide. The streamlines are useful in illustrating the swirl behavior at the surface explicitly. Plus, they help the user to discover an important area where the ideal swirl flow pattern is not being met, namely in the top-center of the geometry just below the intake ports. Here, destructive flow is evident: two regions of flow working against one another. However, caution must be used in the interpretation of this approach. Technically, the velocity has zero magnitude at the surface. What is shown is the velocity just under the surface, extrapolated and projected. No particle actually traverses an entire path shown by a streamline in this case.

Figure 7.5 illustrates streamlines and timelines, used to investigate the behavior of tumble flow at the surface. The streamlines (Figure 7.5 middle-left), again implemented via dye injection as described in Section 7.2, have been seeded in order to selectively highlight a subset of the vector field topology, namely, a *separatrix*–a streamline between two critical points in the vector field [53] or a line that separates two distinct basins of attraction. A saddle, another topological feature, is especially visible between the green and red dye sources. Unfortunately, this is an undesirable feature of the flow for the case of tumble flow.

Timelines are the lines joined by a set of massless particles released into the flow at the same time. The dye-injected emulation of timelines shown in Figure 7.5 middle-right are released from user-specified

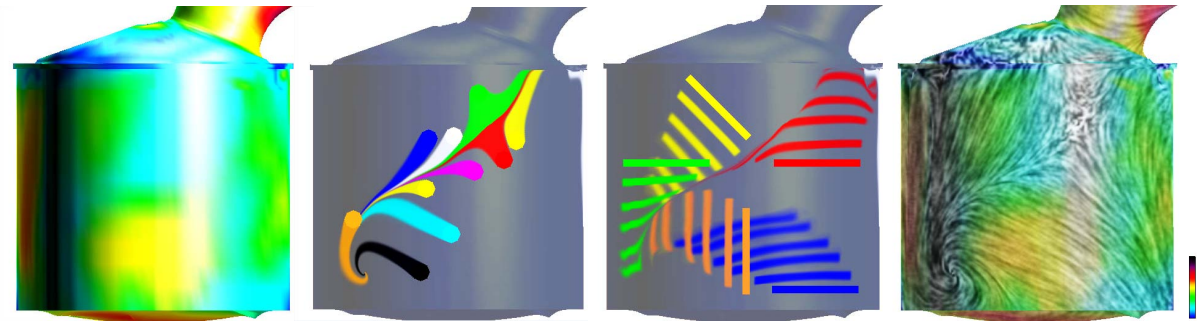


Figure 7.5: The visualization of tumble flow at the boundary surface: (left) direct visualization using color mapping, (middle-left) visualization using dye injection, (middle-right) geometric flow visualization using timelines, and (right) texture-based flow visualization.

locations and orientations. In our implementation, the user may slide, scale, and rotate the release mechanism in addition to specifying color. Allowing the user to specify color is important in order to match correlated timelines. This particular set of timelines is helpful in visualizing convergent (indicated by the yellow and green timelines) and divergent (indicated by the blue timeline) areas of the surface flow. Ideally, the flow emerging from an intake port exhibits divergent behavior.

Texture-Based Flow Visualization

Texture-based visualization is very useful for generating an overview of the flow behavior at the surface. The advantage is that we obtain complete coverage of the flow and maintain spatial coherency according to the flow simultaneously. Plus, flow orientation (upstream vs. downstream flow) may be visible in a still image when a spot noise-like texture is used, although the flow orientation is much clearer in an animated sequence. If we indicate velocity magnitude by the amount of smearing in the texture, as in Figure 7.4 middle-right, than we can encode another CFD attribute as hue such as temperature. On the other hand, it is more difficult to see the properties of the flow in areas of low velocity magnitude. We can address this by applying a velocity clamp as shown in Figure 7.4 right. The result resembles LIC. Again, the destructive flow pattern is very noticeable with a texture-based approach. This is especially clear in a close-up view of the surface. We also note that the interpretation of the results shown in this case may be considered less misleading since long, explicit particle tracing paths are not depicted.

In Figure 7.5 right, we can see that the vortex that characterizes the ideal tumble motion is off-center in the lower-left of the geometry whereas the ideal tumble motion is characterized by a vortex in the center about the z axis pointing towards the viewer. In the next section, we show how to visualize this vortex in 3D.

7.4 3D and Hybrid Approaches

Slices are commonly used to investigate the properties of the flow *inside* the volume but visualizing 3D characteristics of the flow like swirl can be difficult with 2D slices. We are interested in visualization techniques that provide insight into the spatial dimension orthogonal to the slice as well. In this section we discuss 3D and hybrid approaches. In this case, hybrid approaches are both a mixture of spatial dimensions, e.g., 2.5D and 3D as well was a mixture of flow visualization classifications such as geometric and texture-based flow visualization.

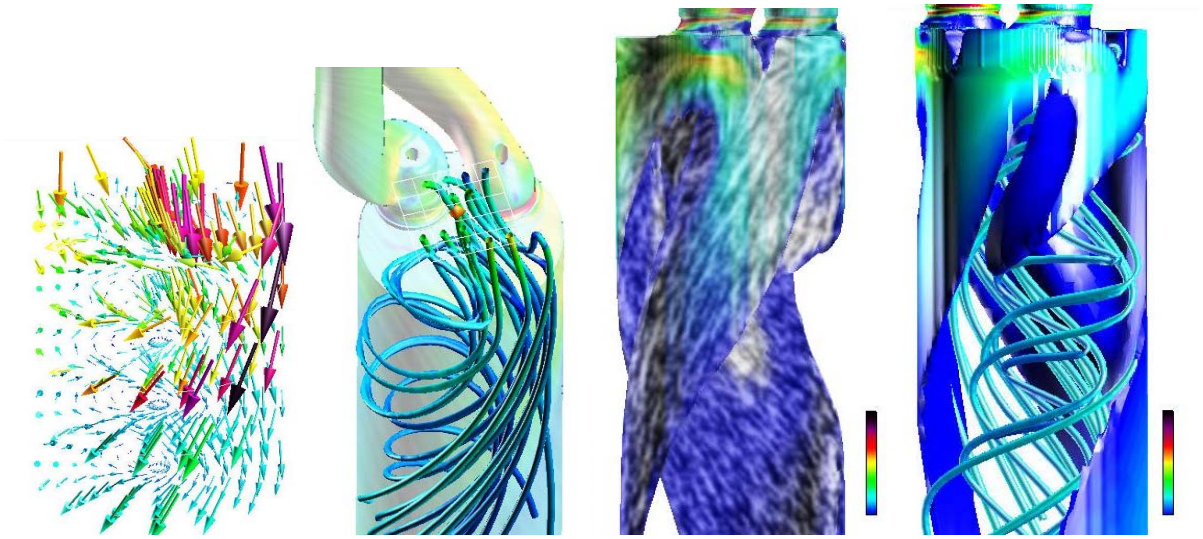


Figure 7.6: The visualization of swirl flow in the volume of the the combustion chamber: (left) direct visualization using color-mapped glyphs, (middle-left) geometric flow visualization using color-mapped shaded streamlines, (middle-right) texture-based flow visualization on a velocity isosurface, and (right) isosurfacing combined with 3D streamlines.

Direct Flow Visualization

Direct flow visualization techniques are generally difficult to apply in 3D. Figure 7.6 left shows 3D swirl flow visualized using color-mapped vector glyphs. This image suffers from perceptual problems such as occlusion, visual complexity, and lack of depth cues. Furthermore, this is not a brute-force hedgehog approach. A subset of the 3D combustion chamber data has been resampled onto a regular polar grid similar to Figure 7.2. Even with the resampling approach, many of the individual glyphs are occluded by larger glyphs or too small to see clearly. The consequence is that we usually view only a subset of the 3D data.

Geometric Flow Visualization

Without special handling, geometric techniques can also suffer from some of the same perceptual problems that direct flow visualization can. One means by which to focus on a particular subset, area of interest, or feature of a flow field is via a streamline seeding strategy. In general, three popular streamline seeding strategies are often used: (1) *image-based* seeding strategies such as that described by Turk and Banks [159] or the evenly spaced-streamline seeding strategy presented by Jobard and Lefer [65], (2) *topological* or feature-based, seeding strategies such as those presented by Verma et al. [169] or Sanna et al. [128], or (3) *interactive* seeding strategies using a streamline seeding rake used by Bryson and Levit [14] or Schultz et al. [137]. Our approach falls into the third category–an interactive streamline seeding strategy.

A schematic of our interactive streamline seeding tool is shown in Figure 6.5. This tool provides the user with six interactive degrees of freedom (DoF): (1-3) three translational, (4) scaling, (5) rotational, and (6) resolution control. These interactive DoFs are required to investigate the results of CFD simulations because the meshes from CFD embrace a wide variety of components, features, and levels of resolution. Ideally, the tools used to analyze and visualize these data sets should be flexible enough to adapt their size, orientation, and resolution to fit the features of interest either automatically or through user-specified

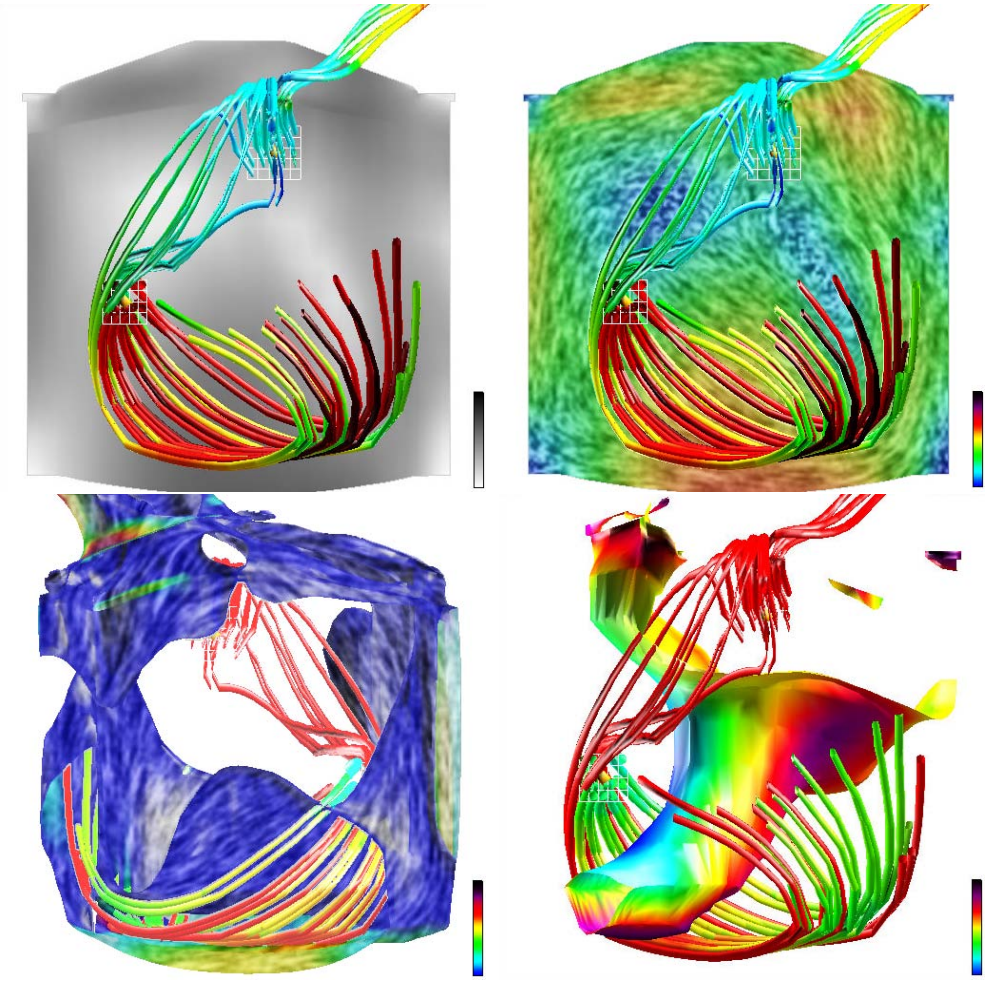


Figure 7.7: The visualization of 3D tumble flow: (top-left) geometric visualization using streamlines, (top-right) geometric visualization streamlines accompanied by texture-based flow visualization on a 2D slice, (bottom-left) geometric flow visualization using timelines, combined with texture-based flow visualization on a velocity isosurface, and (bottom-right) geometric flow visualization with streamlines and a pressure isosurface color-mapped with velocity magnitude.

parameters. Figure 7.6 middle-left, shows streamlines color-mapped with velocity magnitude, placed according to seeding plane in order to visualize the 3D swirl flow. We have employed an approach where a semi-transparent, velocity color-mapped surface serves as context information. In this case, the streamlines show the swirling behavior of the flow rather clearly. The main behavior of the flow appears to match the sought after swirl flow pattern.

Swirl flow is more stable than tumble flow, one reason being that the axis of rotation and the cylinder axis are aligned with swirl motion. In tumble motion, these two axes are orthogonal making it more difficult to realize and visualize. Figure 7.7 top-left shows the visualization of tumble flow with streamlines seeded by two seeding planes. A 2D slice with a gray-scale velocity magnitude color map is used for context. The top-center seeding plane uses streamlines color-mapped according to velocity magnitude. The bottom-left seeding plane uses red streamlines only in order to help distinguish seeding plane-specific streamlines. The bottom-left seeding plane also uses a tool that lets the user interactively control the length of the streamlines. In this case, the tool has been used to truncate the geometry in order to reduce occlusion and visual complexity.

Texture-Based Flow Visualization

In a hybrid visualization approach, we apply ISA for producing dense, texture-based representations of flow on isosurfaces. Isosurfaces are a visualization tool routinely used to investigate the properties of the flow inside a 3D volume. The shape of an isosurface can give us insight into its 3D characteristics.

Figure 7.6 on page 89 right shows two geometric approaches combined, namely, color-mapped streamlines and a velocity isosurface. Applying texture-based flow visualization techniques to isosurfaces provides even more insight into the characteristics of 3D vector fields. This has only recently become a feasible option. Figure 7.6 on page 89 middle-right, shows a velocity isosurface of 5.0 m/s in the combustion chamber of the data set in Figure 5.1 on page 63 with texture-based flow visualization applied. We can see that the flow has some of the swirling orientation that we are looking for. The application of texture advection provides a clear indication of the upstream and downstream nature of the flow that is not visible with an isosurface alone. In particular, the texture-based approaches help to point where the flow does *not* follow the ideal swirl pattern that the combustion chamber should encapsulate, in this case in the top-middle between the two intake ports.

We also note that caution is advised when interpreting the result of texture-based flow visualization on isosurfaces because the direction of the flow does not necessarily align with the isosurface itself. Different approaches to incorporating this normal component of the flow with respect to the isosurface in the visualization include: (1) varying the texture convolution filter [3] or the surface color opacity according to the cross flow [84] or the possibility of not projecting the velocity vectors onto the surface [165]. For a more detailed discussion on this topic we refer to previous literature [84].

Figure 7.9 top-left shows a hybrid visualization using direct color-mapping, streamlines, a velocity isosurface, and texture-based flow visualization. We note that in order to combine the ISA implementation with streamlines, we must ensure a proper rendering order to the objects. All other objects such as streamlines must be rendered before applying texture synthesis so that the image overlay from ISA does not cover these other objects [83]. One perceptual problem with the result in Figure 7.9 top-left on page 94 is occlusion. Figure 7.9 top-right on page 94 illustrates the use of a clipping plane to reveal occluded flow structures such as the isosurface surrounding the intake ports.

Figure 7.7 top-right on page 90 shows a similar visualization with the addition of texture-based flow visualization applied to the slice. In this case, we can see that a subset of the 3D flow does reflect the characteristics of the tumble flow pattern. However, we can see that the axis of flow rotation is off-center. Instead of the axis pointing straight out along the z axis, it is at an angle pointing down and to the left. It looks like a hybrid of the swirl and tumble flow patterns. Figure 7.7 bottom-left on page 90 shows a texture-based flow visualization applied to a velocity isosurface of 7.5 m/s combined with 3D streamlines. In this case we have chosen a view from the back in order to reduce occlusion. The streamlines also verify that the flow exhibits a spatial component normal to the isosurface because they pass through the isosurface near the bottom.

The choice of isovalue is important when using isosurfaces to visualize the flow motion. The users rely on *a priori* knowledge of the data set from CFD experience in addition to some trial and error in order to obtain insightful results. The isosurface in Figure 7.7 bottom-right on page 90 is in fact not optimal because of its shape. The complexity of the isosurface does not lend the user an intuitive interpretation. That is why we chose another approach in Figure 7.7 bottom-right which shows an isosurface of uniform pressure–80,500 Pa (pascals). This result combines two geometric flow visualization methods namely isosurfaces and streamlines. This isosurface helps us to visualize the tumble flow's axis of rotation even more clearly than the streamlines alone. We may choose other CFD attributes from which to compute isosurfaces useful for visualization. Another popular attribute choice is that of *spray equivalence ratio*.

The spray equivalence ratio is made of mass of air over mass of fuel. Thirteen parts of air for each part of fuel equals a spray equivalence ratio of unity. A ratio in the range of 0.7-1.4 represents an ignitable mixture. We use isovalues in this range to help track where the ignitable mixture is located. Figure 7.9 bottom-right on page 94 combines three tools into a hybrid visualization: (a) texture-based flow visualization on a slice, (b) the pressure isosurface color-mapped according to velocity magnitude and, (c) 3D streamlines.

We have also looked at 3D texture-based flow visualization based on a 3D IBFV implementation [153] and a programmable graphics hardware implementation [180] in the context of investigating swirl and tumble flow. We generally find perceptual problems to be a great challenge with this approach. Similar to the resampling approach, a trade-off must be made between coverage of the flow domain and visibility of the flow. This is one reason why a hybrid of geometric and texture-based techniques is useful. Additional challenges for 3D texture-based approaches still remain implementation and efficiency issues because all the simulation results we present are on unstructured, adaptive resolution grids.

7.5 Trade-Offs

We have investigated two typical flow patterns from CFD using three classes of flow visualization techniques commonly available in 2D, 2.5D, and 3D. [2] Our side-by-side comparison of each flow visualization category illustrates that each has its respective advantages and disadvantages. Direct flow visualization techniques are intuitive, easy to implement, common, and insightful. The direct flow visualization techniques are useful in highlighting extremal CFD simulation data values on surfaces. However, direct approaches may not communicate flow evolution very clearly and are often more difficult to apply in 3D. And important features can be missed if the sampling rate for a glyph-based representation is not high enough.

Geometric methods are also intuitive, provide insight, and can be applied to 2D, 2.5D, and 3D vector data. They also sometimes indicate flow direction including the upstream and downstream direction of the flow. Geometric techniques are useful for gaining insight into the location of the axis of rotation for both the swirl and tumble flow patterns while texture-based techniques provided useful enhancements to these results (Figure 7.6 on page 89 and Figure 7.7 on page 90). However, the drawback with these approaches is generally that of placement. Important features may be overlooked because they do not provide complete coverage of the flow domain.

Texture-based approaches share advantages with both direct and geometric techniques by providing complete coverage and showing the direction of the flow everywhere. However, they are difficult to apply in 3D. The geometric and texture-based techniques applied to surfaces were very good at pointing out where the ideal swirl flow pattern was not being met (Figure 7.4 on page 87) in the CFD simulation model. The geometric techniques applied to slices and surfaces were also suitable for highlighting specific topological features of the flow (Figure 7.3 on page 85 and Figure 7.5 on page 88) whereas the texture-based approaches were very helpful by insuring that these topological features were not initially overlooked. Figure 7.8 summarizes some of the trade-offs that are made when visualizing swirl and tumble motion. For example, The more dense the visualization, generally the more difficult is to perceive the result. Thus a trade-off is often made between these two factors. Also a trade-off may is made between spatial coherence and spatial dimensionality because results in 3D often contain many overlapping parts. As a result of these trade-offs, the flexible combinations of approaches offered by our system are good alternatives.

[2] For supplementary material including high resolution images and MPEG animations, please visit:
`http://www.vrvis.at/ar3/pr2/swirl-tumble/`

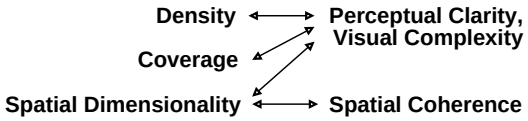


Figure 7.8: In general, trade-offs are made between the density, coverage, and spatial dimensionality of the visualization with that of perceptual clarity, visual complexity, and spatial coherence.

7.6 Discussion and Future Work

In general, the optimal flow visualization technique depends on the needs of the user and the nature of vector field. For example, visualizing swirl flow using 3D streamlines is easier than for the case of tumble motion. We are able to emphasize and communicate different characteristics of the flow with different tools. Also, some methods are better for visualizing 2D flow rather than 3D flow. Hence, a range of tools is required to help us analyze CFD simulation data. Although we have focused on two specific flow patterns from CFD, we believe what we present here to be applicable to more general cases.

Future work includes the application of these three classes of flow visualization techniques to the investigation of motion associated with cooling jackets. The geometry and flow patterns associated with cooling jackets are generally even more complicated than that of swirl and tumble flow within a cylinder. We suspect that techniques which apply to the surface may be especially important since this type of geometry has a very high surface area. Also, since cooling jackets may contain many thin pieces in their geometry, visualization via slicing does not show as much spatial coherency as via surfaces. Three dimensional techniques may also prove to be very useful because cooling jackets are often characterized by components with thin volumes. Hence, texture-based flow visualization techniques applied in 3D may result in fewer perceptual issues.

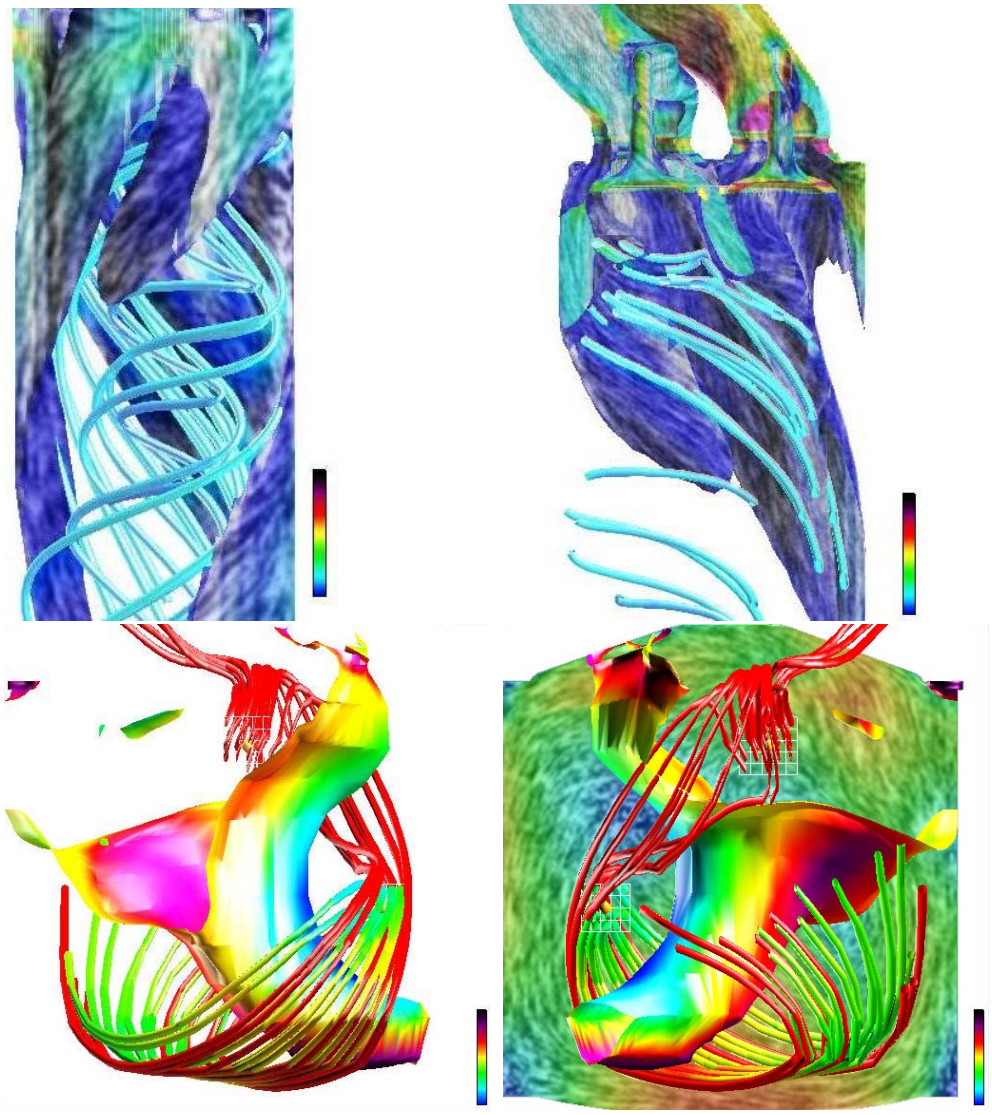


Figure 7.9: Visualization of swirl and tumble flow using a combination of direct color-mapping, streamlines, isosurfaces, texture-based flow visualization and slicing. (top-left) visualizing swirl flow using 3D streamlines and texture-based flow visualization on an isosurface, (top-right) a clipping plane is applied to reveal occluded flow structures, (bottom-left) an isosurface and 3D streamlines visualize tumble motion, and (bottom-right) the addition of texture-based flow visualization on a color-mapped slice.

Chapter 8

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

"All progress is precarious, and the solution of one problem brings us face to face with another problem."

– Martin Luther King Jr. [1] (1929–1968)

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. In our experience, CFD software can be structured according to 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 presented, explored, and analyzed with a variety of visualization tools

The process is iterative, as illustrated in Figure 8.1. 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 8.1. 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 Fig-

[1] US black civil rights leader and clergyman, Nobel Prize in Peace 1964, assassinated

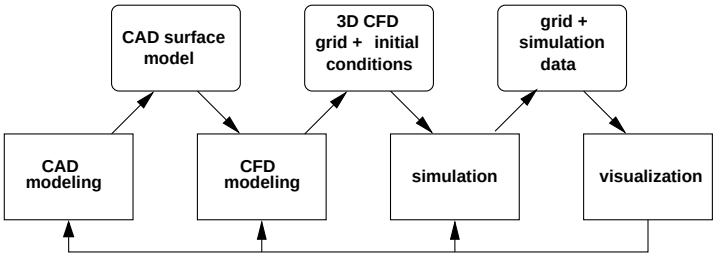


Figure 8.1: The CFD process is iterative and can be pipelined into modeling, simulation, and visualization stages. Rounded boxes represent input/output data while processing stages are depicted as rectangles.

ure 8.1. 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 chapter is organized as follows: Chapter 1 describes the four classes of flow visualization techniques that form the basis of design. Section 8.1 outlines the user requirements and goals of the visualization software. Section 8.2 presents the overall design of the visualization system while Section 8.2 details the implementation and design of our flow visualization software modules. Section 8.4 evaluates some aspects of our design and implementation and discusses some advantages and disadvantages of our work.

Our flow visualization classification is described in the introductory chapter Section 1.2 on page 5. 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.

8.1 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 [57]. 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 1.1 standard.

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.

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 3.1 on page 35 showing two intake ports. Again, looking at an overview, 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 3.2) 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 3.1 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 [171].

8.2 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 8.1 and in more detail in Figure 8.2.

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 [184] 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. [184] is shown in Figure 8.2. 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 8.2 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

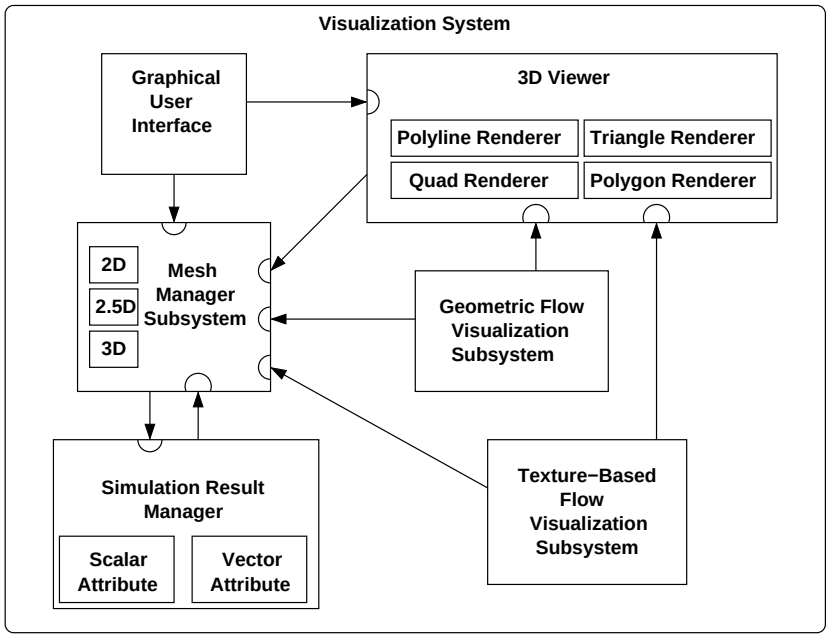


Figure 8.2: A schematic of the design of the visualization system into which we incorporated our research related software. Only the major subsystems are illustrated.

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 were 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.

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.

The Geometric Flow Visualization SubSystem

Figure 8.3 illustrates the main processing pipeline of the geometric flow visualization subsystem. Again, input/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.

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

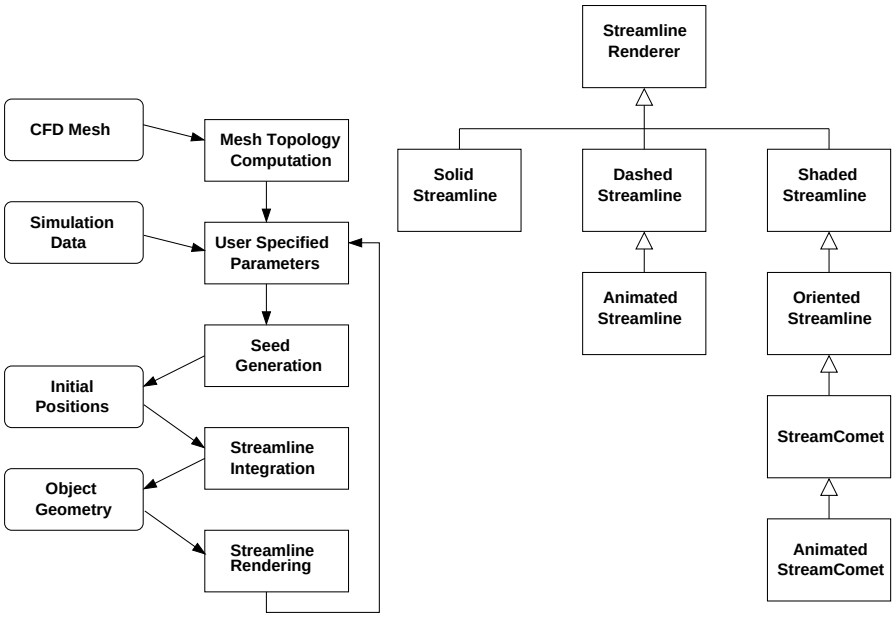


Figure 8.3: (left) The processing pipeline for the geometric flow visualization subsystem. (right) The class hierarchy of streamline and geometric flow visualization options. UML notation is used.

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 8.3 allows higher order integrators such as a second order Runge-Kutta [22] 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.

Figure 8.3 right shows our streamline rendering options displayed in the class hierarchy in which they were designed and implemented. The hierarchy, following UML notation [45], 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.

The class hierarchy contains several child classes. **Solid Streamlines** are rendered as solid OpenGL polylines. **Dashed Streamline** are rendered as dashed lines for reduced occlusion in 3D flow visualization. **Animated Streamlines** are a type of dashed streamline, animated in order to show flow orientation, i.e., upstream and downstream flow direction. **Shaded Streamlines**: are composed of polygons of a finite width in order to facilitate depth perception in 3D. **Oriented Streamlines** are a kind of shaded streamline using semi-transparent polygons in order to convey flow orientation in a still image. A **Streamcomet** is a kind of oriented streamline represented as a glyph with an streamline forming the tail. An **Animated Streamcomet** shows the downstream direction of the flow. 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.

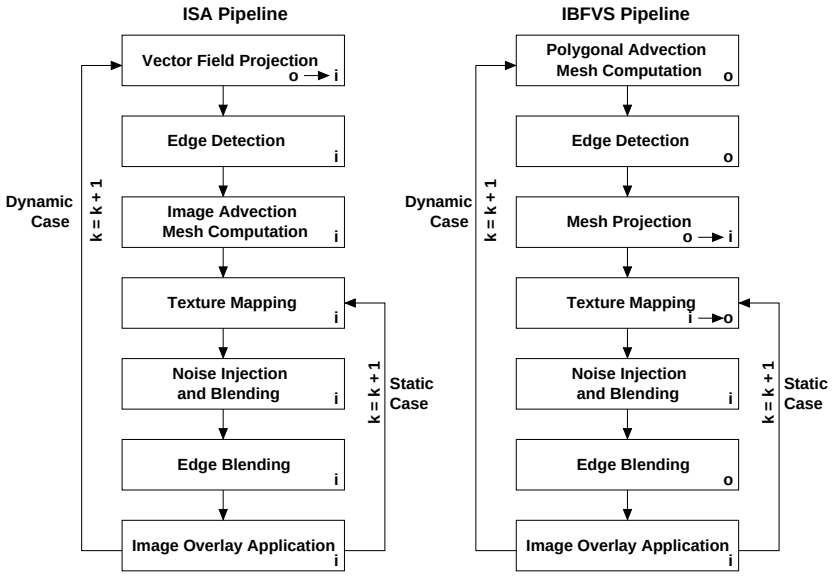


Figure 8.4: The processing pipeline of the texture-based flow visualization subsystem.

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 [77], 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 added 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.

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) [164], Image Space Advection (ISA) [83], and Image Based Flow Visualization for Curved Surfaces (IBFVS) [85, 165]. The ISA and IBFVS algorithms were implemented within the same software package in order to compare them with one another.

A side-by-side illustration of the processing pipelines of both algorithms is shown in Figure 8.4. 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 8.4, 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 Chapter 4 and in previous research [85].

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 8.4, 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 8.4 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 8.5 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 8.3 right we show *composition*, a variation 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 [45]. 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.

Here we outline the major components that make up the texture-based flow visualization subsystem shown in Figure 8.5. The **Texture-Based Flow Visualizer** is the class with the most responsibility, namely that of coordinating pipelines in Figure 8.4 of both the ISA and IBFVS algorithms [85]. 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 [164], ISA [83], and IBFVS [85, 165] algorithms. The **Texture Stack** is composed of individual **Textures**. It is worthy to 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 8.4 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 (Chapter 4 [83]). 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 8.3 and 8.5 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.

8.3 User Interface Design

User interface (UI) design is a classic topic in the field of human-computer interaction. UI Design professionals are constantly seeking ways to ensure that software products are developed with the end-

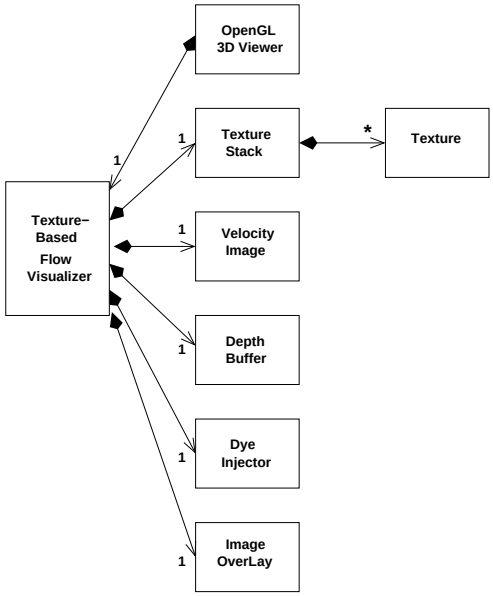


Figure 8.5: The major components of the texture-based flow visualization design. Here aggregation, or *is-part-of*, relationships are shown.

user's goals in mind. The goal being to make them more powerful, to facilitate the users' job, and to make the software enjoyable for the people who use it. Our application targets the community of engineers that use the software for CFD. One key idea this GUI design stems from the need of the users to compare different data sets to one another. In the field of CFD, engineers first alter design parameters of their models or initial conditions, then run a simulation, and finish by comparing the results with previous trials (Figure 8.1). This cycle is carried out in order to judge whether or not the results have improved. Thus, from a user-standpoint, the UI is required to explicitly support this comparative analysis and visualization scenario.

Although our application is fairly specialized, we believe the principles outlined can be applied to a broad spectrum of applications such as software tools for power train design (the mechanism by which power is transmitted from an engine to a propeller or axle), acoustic simulation, hydraulic systems, piston design and simulation, as well as various applications related to thermodynamics.

The amount of literature dedicated to the subject of UI design is vast [23, 133, 155]. An exhaustive review of the topic is beyond the scope of this chapter however, we mention some influential work here. In general, the work of Cooper and Reimann [23], Nielson [107], and Shneiderman [133] is certainly very influential. The guidelines they provide are invaluable in terms of our daily work.

We also note recent, specialized, related literature. Beier and Vaughan present important UI guidelines for web applications [6], a very popular area of focus in recent years. Epstein and Beu describe another industry-level user interface design for process control [40]. We also see UI design guidelines for common house-hold appliances such as televisions [20].

The User Community

The design of the UI (discussed in detail in the following section) is a result of continuous feedback from our users. In this case, the user community is a rather focused group of engineers. The educational and professional background of the users spans multiple disciplines including electrical engineering, automotive engineering, application engineering, and even business. But the majority of the users are

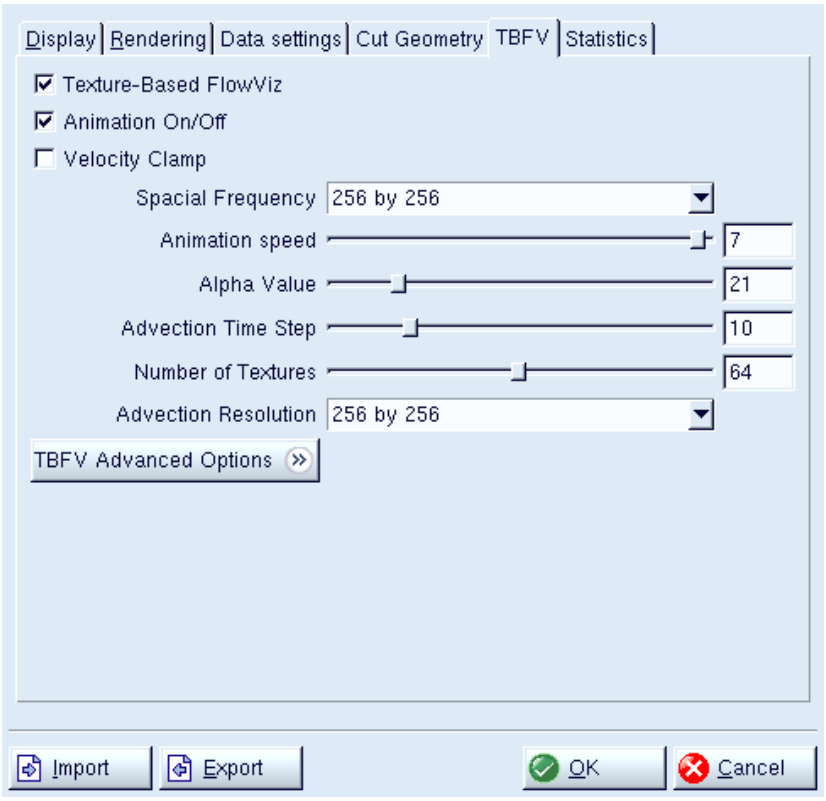


Figure 8.6: Here we see a snapshot of a UI dialog that provides explicit support for texture-based flow visualization and comparative analysis of CFD simulation data.

mechanical engineers. They therefore have a higher level of training and background knowledge than the average user. This can be considered a requirement in this context. Working in close contact with this group of users allows us to incorporate feedback directly from the same individuals using the software on a daily basis as part of their full-time or part-time occupation.

Graphical User Interface

Before going into a detailed discussion the individual features in the UI, we describe some of the higher level components that whose purpose is to explicitly facilitate and support flow visualization and comparative analysis. Figure 8.6 shows a snap shot from the design. The appearance is simple and the components consist of familiar widgets including check-boxes, drop-down menus, sliders, text-boxes, and buttons. The application scenario is the visualization of CFD simulation data, specifically the texture-based flow visualization module described in Section 8.2. We show a specific visualization example in Figure 4.13. Figure 4.13 shows the visualization of flow at the surface of an intake port and piston valve (more details in the next section). This data set is the result of running a CFD simulation on the model of a piston valve. The engineers are interested to see how closely the characteristics of the flow match the ideal tumble [86] flow pattern they are trying to achieve.

In summary, the basic UI dialog components from Figure 8.6 are:

- *check-boxes*: The check-boxes at the top are there to support boolean options that enable and disable different flow visualization options such texture-based flow visualization, animation, and velocity vector clamping.

- *drop-down menus*: These supply a range of fixed-valued options to the user. Not all the parameters are good candidates for a continuous range of values.
- *sliders*: The sliders enable the user to interactively search a wide range of values relevant to the analysis and visualization
- *text-boxes*: The text-boxes are automatically updated to reflect their neighboring slider values (and vice-versa). Also important is that the user may enter precise values into the text boxes, necessary for comparing different data sets.
- *buttons*: Standard buttons are used to enable and disable the dialog as well as import and export user settings.

The *Import* and *Export* buttons at the bottom, left of the dialog are an important feature of the design. A user may spend hours fine tuning the parameters contained in a dialog to suit the resulting data set and to suit their individual needs. Therefore, they need to save each of these dialog settings (using the *Export* settings button) for a minimum of two reasons. Firstly the users must be able to stop and re-start their analysis and visualization sessions. Since these sessions may last several hours, they are not necessarily completed in one sitting. And secondly (the more important reason), the users must be able to save their settings, off-load the current CFD simulation data, re-load a new set of simulation results, and then return to the exact same set of user specified parameters, in this case, by using the *Import* settings button. As mentioned previously, engineers alter the design parameters of their models or initial conditions and then run another simulation. A complete CFD cycle starting with model creation, running a simulation, and inspecting the results can take several weeks. Persistent storage of dialog settings is required for the comparative analysis of the CFD simulation results.

If the user chooses to export the dialog settings to a text file, then the settings can also be easily shared with other users. For example, if an engineer discovers something important that they think another remote user should see, they can export their dialog settings and simply send them via email to another user. The other user simply needs to start up the application and import the settings.

Detailed Feature Description

We continue our discussion with a description of the individual features contained in the prototype dialog shown in Figure 8.6 on page 103. In this example, a preliminary analysis is done via flow visualization as shown in Figure 4.13 on page 61. Visualization can be used to convey important characteristics of flow through a model and can help the engineers see if the behavior follows the sought after pattern. Texture-based flow visualization techniques like the one shown here have the advantage of providing complete coverage of the flow field. Only recently have these techniques become fast enough to support interactive exploration.

The first check box (Figure 8.6 top, left) enables and disables the textures that produce the visualization. In this case, texture patterns are smeared in the direction of the vector field. Longer streaks indicate higher velocity, while shorter streaks indicate lower velocity. The animation check-box allows the user to freeze the current imagery (stop the animation) in order to gain a static picture of the vector field. This is useful for taking snap shots for presentations. The velocity clamp check-box allows the user to clamp the velocity vectors with a high magnitude to a fixed maximum. When this option is used, the texture-streaks tend to converge to a uniform length. This option can help in visualizing critical points, e.g., sources and sinks, in the flow.

The Spatial Frequency drop-down menu allows the user to change the spatial frequency of the noise patterns used to generate the smeared textures. A higher spatial frequency shows more detail while lower frequencies are more suited for generating longer streak patterns.

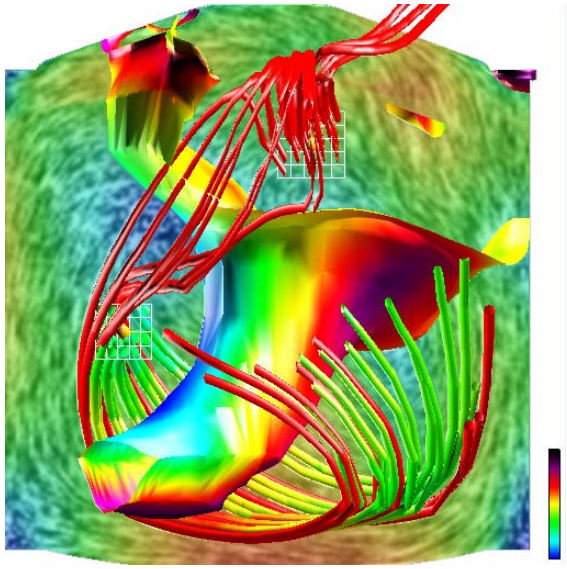


Figure 8.7: Visualization of tumble motion using a combination of several visualization options including: texture-based flow visualization, isosurfacing, streamlines, and color-mapping.

The Animation Speed slider controls the frame rate of the animation. The Alpha Value slider controls the opacity level of the textures. A higher opacity makes the texture pattern more perceivable to the eye while a lower opacity increases the visibility of the underlying shaded and colored surface. The Advection Time Step slider influences the distance that a texture property can be transported in one animated frame. The term *advection* refers to the movement of a mass of fluid or the transport resulting from such movement. A shorter advection time slows down the flow of texture properties (as a global setting) while a longer advection time increases the distance they may travel from one animated frame to another.

The slider controlling the number of textures is tailored for the wide variety of graphics cards coupled with the users desktop PCs. Some uses have older machines, with older graphics cards containing less texture memory. For these machines, the number of textures used should be smaller. Users with modern machines can use a larger number of textures and thus generate higher quality imagery.

The Advection Resolution drop-down menu also influences the quality of the results. A higher resolution generates higher quality imagery, but requires longer computation time thus reducing the frame rate. A lower resolution results in higher frame rates, but is less slightly less accurate. Therefore, we recommend the user do their initial exploration with a lower advection resolution, and use the highest resolution for presentation or when the highest accuracy is required.

Finally, some advanced options such as dye injection can be brought up in another dialog with the same design. Although these user options may seem rather complicated at first sight, their effects become more obvious in an interactive setting where the user modifies a parameter and immediately sees the consequences in the resulting visualization. Recall also that the target users in this case are typically engineers with experience in CFD. Note that this description of the GUI pre-dates the implementation of the IBFVS algorithm.

8.4 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 8.1.

A big advantage of integrating the research related subsystems into a larger commercial system is the ability to combine visualization options. Figure 8.7 on page 105 shows the visualization of tumble flow [86] using a combination of texture-based flow visualization, color-mapping, streamlines seeded with two seeding planes, and a color-mapped pressure isosurface [2]. Figure 8.8 on page 108 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 8.8), 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. In terms of versatility, our tools, having been incorporated into commercial software, must undergo more testing 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. Our research modules have also been tested by more users than typical prototype software. 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 techniques will be applied.

[2] For supplementary, high resolution images, please visit:
`http://www.VRVis.at/ar3/pr2/design/`

Our range of tools also includes those that address the perceptual challenges in 3D visualization. These tools include the streamrunner [77], streamcomets, and variable resolution streamline seeding plane [86]. However, applying texture-based flow visualization techniques to true 3D flow still remains an unsolved problem in this context.

We have introduced a GUI design prototype with explicit support for geometric and texture-based flow visualization and comparative analysis. The design is driven by the users' need to perform a specialized set of operations in order to carefully compare multiple data sets resulting from CFD simulations. We believe this UI design to be a valuable asset in helping a broader user community perform their daily tasks optimize their results.

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.

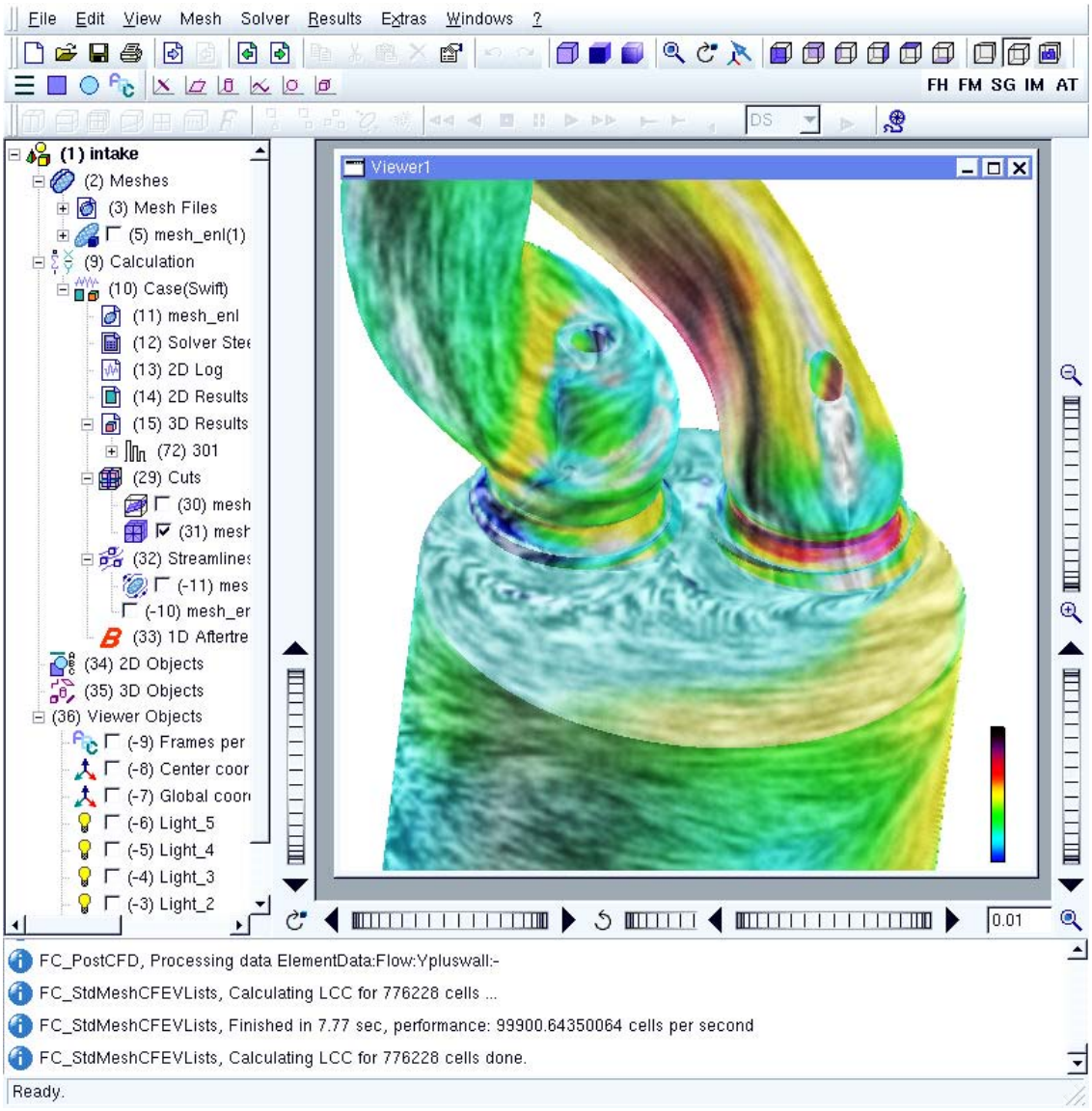


Figure 8.8: A screen shot of our industry level application being used to visualize the vector field at the surface of two intake ports.

Chapter 9

Summary

"It's a job that's never started that takes the longest to finish."

– J. R. R. Tolkien[1] (1892–1973)

Visualization is an important part of exploring, analyzing, and presenting the results of a CFD simulation. As the size of CFD simulation data sets increases, the utility of scientific visualization for gaining insight into the data sets also increases. Visualization offers one way to manage such large collections of simulation data since it brings the data to a higher level of abstraction. Simply reading the raw data does not meet all of the demands set forth by the user and may not even be feasible. Furthermore, no single visualization solution can span the range of each engineer's needs. Hence a range of solutions must be at the user's disposal.

We present a selection of recent advances in flow visualization that addresses the growing demand for solutions that offer insight into the continuously expanding CFD simulation data sets. Our presentation draws upon flow visualization techniques being classified into four main categories: (1) direct, (2) texture-based, (3) geometric, and (4) feature-based [82, 116]. This classification provides a framework useful for placing our techniques in a larger context. The resampling tool we implemented and present falls into the direct flow visualization category (Chapter 3 [78]). The Image Space Advection (ISA) algorithm we implemented fall into the texture-based category (Chapter 4 [85]). ISA applied to isosurfaces contains elements from both texture-based and geometric flow visualization methodology (Chapter 5 [84]). And our implementation and discussion of geometric techniques includes, oriented streamlines, animated-dashed streamlines, streamlets, and streamcomets (Chapter 6 [80]). These recent advances are applied to real-world applications in the field of CFD including the investigation and visualization of patterns of flow motion specific to automotive engineering. We discuss the benefits of these techniques along the way using real-world results.

The rest of this chapter is organized as follows: Section 9.1 summarizes a resampling tool for CFD simulation data. Section 9.2 summarizes recent algorithms for the fast advection of textures on surfaces. Section 9.3 outlines how texture-based flow visualization can be applied to isosurfaces. Section 9.4 summarizes a collection of geometric flow visualization techniques applicable to CFD simulation data and Section 9.5 mentions how direct, geometric and texture-based flow visualization techniques can be used to solve real-world problems from the automotive industry. We note that much of this material has also been published elsewhere [81].

[1] British (South African-born) Anglo-Saxon scholar and fantasy novelist, author of "The Hobbit" and "The Lord of the Rings" trilogy

9.1 Resampling of CFD Simulation Data

To start off, we introduce a flexible, variable resolution tool for interactive resampling of computational fluid dynamics (CFD) simulation data on unstructured grids. The tool and coupled algorithm afford users precise control of glyph placement during vector field visualization via six interactive degrees of freedom. The resampling tool, called FIRST (a Flexible and Interactive ReSampling Tool), is a valuable asset in the engineer's pursuit of understanding and visualizing the underlying flow field in CFD simulation results.

FIRST solves both the perceptual problems resulting from a brute force hedgehog visualization approach, where a vector glyph is rendered at every CFD grid cell, and glyph placement problems by (1) giving the user control of the *resolution* of the glyphs in the image and (2) giving the user precise control of *where* to place the vector glyphs for viewing the flow with normal components.

Interactive Visualization and Analysis

The key distinguishing features of FIRST stem from the fact that it was specifically developed in order to provide the user with a range of flexible interactions at multiple resolutions. The reason we focus on a combination of user control with resampling is because engineers require interactive visualization solutions. This is partly due to a large amount of time engineers spend searching the data sets. 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 previous simulation results and with measured, experimental results.

FIRST provides the following features: (1) several interactive DoFs: three translational, scaling, rotation, and resolution (Figure 3.6), (2) handles changes to both underlying topology and geometry, i.e., can be utilized for the display of time-dependent, unstructured grid slices where geometry and topology change over time or space, (3) resamples any unstructured grid onto any structured grid, (4) handles unstructured grids with holes and discontinuities, (5) does not rely on any pre-processing of the data, (6) consists of a straightforward implementation, e.g., requires no neighbor-finding capabilities or complicated data structures, (7) processes large quantities of unstructured, scalene triangles efficiently. The resampler provides flexible user-interaction capabilities beyond those offered by other methods. Also, the underlying algorithm operates on a per-unstructured-polygon basis, making it suitable for parallelization.

Resampling Options

The resampler features are associated with a user-defined, 2D slice through a 3D mesh from CFD. Engineers take a slice of the data and slide the slice through the geometry in order to find features of the simulation data, e.g., areas of extreme pressure and vortices. As the user moves the slice through the 3D mesh, the resampler automatically resizes itself around the slice boundary, handling changes to both the underlying geometry and topology. This is important with respect to addressing the versatility of our CFD simulation data sets. Furthermore, requiring the user to manually adjust the size of the resampling grid would slow down the visualization and analysis process considerably.

Figure 3.10 on page 46 illustrates our technique on a mesh with discontinuities. The discontinuities are two gaps in the shape of rings. Visualization of flow with normal components is shown using both the hedgehog technique versus the glyphs onto a resampled grid. [2] Figure 3.10 on page 46 illustrates how FIRST reduces occlusion and visual complexity thus making the results more suitable for presentation. Also, rendering times are accelerated because the number vector glyphs is reduced.

[2] For supplementary images and MPEG animations of the resampler, please visit:
`http://www.VRVis.at/ar3/pr2/resampler/`

9.2 ISA: Image Space Based Visualization of Unsteady Flow on Surfaces

Dense, texture-based, unsteady flow visualization on surfaces has remained an elusive problem since the introduction of texture-based flow visualization algorithms themselves. The class of fluid flow visualization techniques that generate dense representations based on textures started with the Spot Noise [163] and LIC [17]. The main advantage of this class of algorithms is their *complete* depiction of the flow field while their primary drawback is, in general, the computational time required to generate the results.

Here we summarize a new algorithm, ISA (Image Space Advection), that generates dense representations of arbitrary fluid flow on complex, non-parameterized surfaces, more specifically, surfaces from computational fluid dynamics (CFD). However, the algorithm is general enough to apply to other vector field data associated with a surface such as blood vessel flow.

Traditional visualization of boundary flow using texture mapping first maps one or more 2D textures to a surface geometry defined in 3D space. The textured geometry is then rendered to image space. Here, we alter the classic order of operations. First we project the surface geometry to image space and then apply texturing. In other words, conceptually texture properties are advected on boundary surfaces in 3D but in fact our algorithm realizes texture advection solely in image space. The result is a versatile visualization technique with the following characteristics:

- generates a dense representation of unsteady flow on surfaces
- visualizes flow on complex surfaces composed of polygons whose number is on the order of 200,000 or more
- visualizes flow on dynamic meshes with time-dependent geometry and topology
- visualizes flow independent of the surface mesh's complexity and resolution
- supports user-interaction such as rotation, translation, and zooming always maintaining a constant, high spatial resolution
- the technique is fast, realizing up to 20 frames per second

The performance is due, among other reasons, to the exploitation of graphics hardware features and utilization of frame-to-frame coherency.

Physical Space vs. Parameter Space vs. Image Space

One approach to advecting texture properties on surfaces is via the use of a parameterization, a topic that has been studied *ad nauseam* (e.g., Levy et al. [87]). According to Stalling [143], applying LIC to surfaces becomes particularly easy when the *whole* surface can be parameterized globally in two dimensions, e.g., in the manner of Forssell and Cohen [43, 44]. However, there are drawbacks to this approach. Texture distortions are introduced by the mapping between parameter space and physical space and, more importantly, for a large number of surfaces, no global parameterization is available such as isosurfaces from marching cubes and most unstructured surface meshes resulting from CFD. Surface meshes from CFD may consist of smoothly joined parametric patches, but can have a complex topology and therefore, in general, cannot be parameterized globally. Figures 4.2 on page 49 and 4.3 on page 50 are examples of surfaces for which a global parameterization is not easily derived.

Another approach to advecting texture properties on surfaces would be to immerse the mesh into a 3D texture, then the texture properties could be advected directly according to the 3D vector field. This would have the advantages of simplifying the mapping between texture and physical space and would result in no distortion of the texture. However, this visualization would be limited to the maximum resolution of the 3D texture, thus causing problems with zooming. Also, this approach would not be very efficient in that most of the texels are not used. The amount of texture memory required would

also exceed that available on our graphics card, e.g., we would need approximately 500MB of texture memory if we use 4 bytes per texel and a 512^3 resolution texture.

Can the problem be reduced to two dimensions? The surface patches can be packed into texture space via a triangle packing algorithm in the manner described by Stalling [143]. However, the packing problem becomes complex since our CFD meshes are composed of many scalene triangles as opposed to the equilateral and isosceles triangles often found in computational geometry. The problem of packing scalene triangles has been studied by Carr et al. [19]. For CFD meshes, triangles generally have very disparate sizes. For a given texture resolution, many triangles would have to be packed that cover less than one texel. To by-pass this, the surfaces could be divided into several patches which could be stored into a texture atlas [87]. In any case, computation time would be spent generating texels which cover polygons hidden from the current point of view. The preceding discussion lead us to an alternative solution that, ideally, has the following characteristics: works in image space, efficiently handles large numbers of surface polygons, spends no extra computation time on occluded polygons, does not spend computation time on polygons covering less than a pixel, and supports user interaction such as zooming, translation, and rotation.

Method Overview

The algorithm summarized here simplifies the problem by confining the advection of texture properties to image space. We project the surface geometry to image space and then apply a series of textures. This order of operations eliminates portions of the surface hidden from the viewer. In short, our proposed method for visualization of flow on surfaces is comprised of the following procedure:

1. associate the 3D flow data with the polygons at the boundary surface i.e., a velocity vector is stored at each polygon vertex of the surface
2. project the surface and its vector field onto the image plane
3. identify geometric discontinuities
4. advect texture properties according to the vector field in image space
5. inject and blend noise
6. apply additional blending along the geometric discontinuities previously identified
7. overlay all optional visualization cues such as showing a semi-transparent representation of the surface with shading

These stages are depicted schematically in Figure 4.4 on page 51. Each step of the pipeline is necessary for the dynamic cases of unsteady flow, time-dependent geometry, rotation, translation, and scaling, and only a subset is needed for the static cases involving steady-state flow and no changes to the view-point. We consider each of these stages in more detail in Chapter 4.

Discussion

The ISA algorithm supports visualization of flow on arbitrary surfaces at up to 60 FPS via the careful use of graphics hardware. It supports exploration and visualization of flow on large, unstructured polygonal meshes, and on time-dependent meshes with dynamic geometry and topology. The method generates dense representations of time-dependent vector fields building on both the LEA and IBFV algorithms. It also does not waste computation time on occluded polygons or polygons covering less than one pixel. While the vector fields are defined in 3D and associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space.

Future work can go in many directions including visualization of unsteady 3D flow, something we expect to see soon. Challenges will include both interactive performance time and perceptual issues. Future

work also includes the application of more specialized graphics hardware features like programmable per-pixel operations in the manner of Weiskopf et al. [179, 181] and the use of pixel textures like Heidrich et al. [52]. We also note that we have done a thorough comparison of ISA and IBFVS [85].

9.3 Texture Based Visualization of Flow on Isosurfaces

For many of the automotive components that undergo evaluation, there is an ideal pattern of flow that engineers try to create. Figure 5.1 on page 63 illustrates the swirl motion of fluid flow in a combustion chamber from a diesel engine. In order to generate swirl motion, fluid enters the combustion chamber from the intake ports. Later on in the engine cycle, the kinetic energy associated with this swirl motion is used to generate turbulence for mixing of fresh oxygen into the fluid. The more turbulence generated, the better the mixture of air and diesel fuel, and thus the better the combustion itself. Ideally, enough turbulent mixing is generated such that 100% of the fuel is burned.

Since it is the swirling flow that is used to generate turbulence, the swirl should be maximized in order to maximize turbulence. From the point of view of the mechanical engineers designing the intake ports, increased swirl flow leads to beneficial conditions: (1) improved mixture preparation, i.e., more fuel contact with oxygen, (2) a higher EGR (Exhaust Gas Ratio) which means a decrease in fuel consumption, and (3) lower emissions. However, too much swirl displaces the flame used to ignite the fuel. As such, a balance must be achieved between generating enough swirl flow in order to create turbulence and not displacing the flame used to ignite the flow.

Some routine questions that a mechanical engineer may ask when investigating swirl flow are: Can visualization provide insight into or verify the characteristic shape(s) or behavior of the flow? What tool(s) can help to visualize the swirl flow pattern? and Where in the combustion chamber is the swirl flow pattern *not* being met?

Isosurfaces are a visualization tool used routinely by mechanical engineers to investigate the properties of the flow inside a 3D volume. The shape of an isosurface can give the engineer insight into its 3D characteristics. Figure 5.5 on page 67, left shows a velocity isosurface in the combustion chamber of the data set in Figure 5.1 on page 63. The engineer can see that the flow has some of the swirling orientation that they are looking for. However, what is missing from Figure 5.5, left, is a clear indication of flow direction, e.g., the upstream and downstream nature of the flow. In particular, it is not obvious where the flow does *not* follow the ideal swirl pattern that the combustion chamber should encapsulate.

Applying Texture-Based Flow Visualization

Applying texture-based flow visualization techniques to such isosurfaces provides engineers even more insight into the characteristics of 3D vector fields. And this has become a feasible option only recently. We applied the ISA method described in Chapter 4 [83] for producing dense, texture-based representations of flow on isosurfaces. The result is a combination of two well known scientific visualization techniques, namely iso-surfacing and texture-based flow visualization, into a useful hybrid approach. Our application is a versatile visualization technique with the following characteristics: (1) generates a dense representation of flow on adaptive resolution isosurfaces, (2) visualizes flow on complex isosurfaces composed of polygons whose number is on the order of 200,000 or more, (3) visualizes flow independent of the isosurface mesh's complexity and resolution, (4) supports user-interaction such as rotation, translation, and zooming always maintaining a constant, high spatial resolution, and (5) produces fast animations, realizing up to 60 frames per second.

Applying a Normal Mask

When visualizing flow on normal boundary surfaces the direction of the flow generally coincides with the surface itself. As the flow approaches the boundary, it is not allowed to pass through and is pushed in a tangential direction, i.e., it can be described as surface aligned flow. However, in the case of isosurfaces this is no longer true. The flow at an isosurface can sometimes exhibit a strong flow that is normal to the surface, e.g., cross-surface flow. The same also holds true for the case of arbitrary clipping geometries. Simply advecting texture properties according to the vector field projected onto the isosurface could be considered misleading.

Battke et al [3], who applied LIC to surfaces, address this problem by varying the length of the convolution filter according to the magnitude of the vector component tangential to the surface. In areas where the vector field is oriented almost perpendicular to the surface only very little smearing of the texture occurs, i.e., the input noise is visible instead of a convolved texture. Our approach is required to be consistent with the visualization of flow on boundary surfaces. When we apply texture-based flow visualization to boundary surfaces, the amount of texture-smearing indicates velocity magnitude, i.e., texture is smeared into longer streaks in areas of higher velocity magnitude. We don't want to change the semantic interpretation of smearing for isosurfaces.

We propose an idea inspired by the well known velocity mask [64], namely, a *normal mask*. A velocity mask can be used to dim or highlight high frequency noise in low velocity regions. Whereas, a normal mask can be used to dim regions of the vector field that have strong cross-flow component to the isosurface. We define the normal mask as:

$$\beta = (\mathbf{v} \cdot \mathbf{n})^m \tag{9.1}$$

where β increases as a function of the product of the velocity, $\mathbf{v}$, and normal vector to the surface, $\mathbf{n}$, at that point. Here, m is arbitrary. In practice, m is typically around unity giving the opacity a linear behavior. In our case, the image overlay becomes more opaque in regions with a strong cross-flow component and more transparent in areas of highly tangential velocity. With the normal mask enabled, the viewer's attention is drawn away from areas of strong cross-flow component, and towards areas of high tangential velocity. However, the texture properties are still advected according the velocity vectors projected onto the isosurface.

Some results of applying this normal mask to an isosurface are shown in Figure 5.5, right. We can see that the flow at the isosurface just below the intake port in the foreground (in white) has a strong normal component to the isosurface. The higher frequency texture in this region is difficult to see. Note also that we have chosen a simpler color scale in this case to reduce the visual complexity of the result. We find that using a full range of hue for the color mapping in combination with variable opacity for the normal mask is visually complex. So we provide the option of trading off some complexity in the color map while applying the normal mask [3].

9.4 Geometric Flow Visualization Techniques

This section turns our attention to a collection of geometric flow visualization techniques including oriented streamlines, streamlets, a streamrunner tool, streamcomets, and a real-time animated streamline technique. We place special emphasis on necessary measures required in order for geometric techniques to be applicable to real-world data sets. There has been a lot of work done in this area. And while some of the geometric techniques here have been presented in previous literature, they are often not illustrated

[3] For supplementary images and animations of texture-based flow visualization on isosurfaces, please visit:
`http://www.VRVis.at/ar3/pr2/VisSym04/`

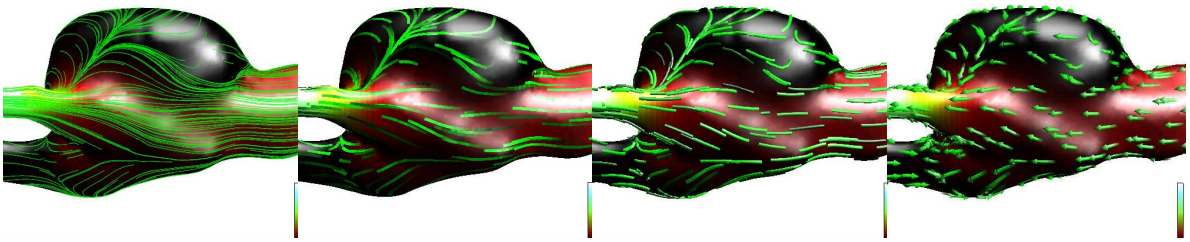


Figure 9.1: The visualization of blood flow at the surface of an aneurysm: (left) geometric flow visualization using streamlines (middle-left) oriented streamlines and (middle-right) streamlets, and (right) streamcomets.

in the context of real-world data sets. For full overview of related research, see the work by Post et al. [116].

Oriented Streamlines, Streamlets, and Streamcomets

One of the drawbacks of conventional streamlines is the lack of flow orientation (upstream vs. downstream direction) depicted in a still image. Our system incorporates an oriented streamline implementation. Oriented streamlines convey the downstream direction of the flow by varying the opacity as a function of particle trace evolution. In other words, the further downstream an integration path is traced, the higher the opacity of the streamline. This can be implemented by giving the streamlines a finite width, either automatically or through user-defined parameters, and using semi-transparent polygons in order to depict an oriented streamline (Figure 9.1, middle-left). Arrow heads could also be used to achieve the same effect. However, arrow head glyphs can more easily lead to visual clutter without careful treatment. The result is similar to that of OLIC (Oriented Line Integral Convolution) [175, 176]. One important difference is that OLIC is based on a traditionally slower approach derived from LIC and OLIC is more applicable to 2D flow rather than 3D. Another method for addressing the speed issue was introduced by Löffelmann et al. however it was applied to 2D flow only [92].

For the case of unsteady flow, drawing a continuous particle path using only a single time step can be considered misleading. This is because no particle actually traces such a path. For the case of slices and surfaces, the visualization becomes even more problematic because a component of the vector field is taken away, namely that component orthogonal to the slice or surface, absent after a projection onto the slice or surface. One approach to handling this is through the use of streamlets (short streamlines) as illustrated in Figure 9.1 middle-right.

Streamcomets are an extension of the streamrunner [77]. The streamrunner addresses the problems of occlusion and scene complexity directly by giving the user control over the evolution of streamlines from seeding time until they terminate. A streamline may terminate when it reaches a boundary in the geometry, reaches a region of zero velocity, or reaches a maximum length set by the user. Streamcomets follow a very intuitive metaphor. They offer four interactive degrees of freedom as shown in Figure 6.8 on page 78. Coupled with more interactive degrees of freedom, streamcomets offer the advantage of showing local flow direction and curvature for static images. There is strong evidence to support the notion that flow visualization objects that show the direction of the local vector field improve the user's ability to identify critical points and understand particle advection paths [75].

Another useful feature is the option of animating the streamcomets. Conceptually, animating the streamcomets such that the comet head position is automatically incremented along the streamline path, acts as a visual search function. The viewer is able to use the animation to search for optimal comet head posi-

tions. This is very useful when the user is not sure where to position the head, searching for interesting features in the flow field, or optimizing the other interactive DoFs. We emphasize the importance of the user's ability to resize the streamcomets along arbitrary dimensions when zooming in and out of the data sets. Changes to the diameter of the comet heads apply to the entire collection of streamcomets, and are not applied on a per-comet basis. Applying size changes to individual comets would lead to misleading visualization results, e.g., the user may interpret different comet head sizes to be a reflection of scalar properties inherent in the flow field.

Figure 9.1, left-to-right, shows the use of streamlines, oriented streamlines, streamlets and streamcomets all applied to the same data set. The data set in this case is simulation data coming from blood flow through an aneurysm. Note that these techniques highlight flow characteristics such as areas of divergence and convergence. Furthermore, they are well suited for 3D flow visualization of which we will see more in Section 9.4.

Animated Streamlines

We use a stippling approach to animate streamlines such that the downstream direction of the flow is depicted. The advantage here is that the stippling approach is supported by OpenGL 1.1 and commodity graphics hardware. Thus real-time frame rates can be achieved even for large numbers of streamlines. Anti-aliasing, also supported by the graphics hardware, can be added to visually enhance the results at very little overhead. Our approach is reminiscent of that used by Jobard and Lefer [66] or Berger and Gröller al. [7] where a color-table look-up approach is used to animate the streamlines. One important difference is that the technique here applies well to 3D flow.

Figure 6.6 on page 76 shows shaded streamlines and animated–dashed streamlines used to visualize tumble motion [86]. [4] The sparser animated–dashed streamlines allow the user to see through the volume and line stipple patterns can be rendered at fast frame rates. See Chapter 6 [80] for more details on real-time frame rates. The implementation is simpler than the dash tube technique of Fuhrmann and Gröller [47].

9.5 Investigating Swirl and Tumble Motion

The VRVis Research Center collaborates with AVL (www.avl.com) in order to provide visualization solutions for analysis of their CFD simulation results. Previously, AVL engineers used a series of color-mapped slices to assess and visualize the results of their CFD simulations. Isosurfaces were used less commonly to assess certain 3D features that could not be investigated sufficiently with 2D slices. Recently, new solutions for the visualization of CFD simulation data have been introduced. This section reports on the application of these techniques in addition to the more traditional approaches. We describe: (a) the application of different visualization techniques to specific application cases, (b) advantages and disadvantages of what these techniques offer, and (c) a comparison which may apply to other application cases. We also to give recommendations on when to use specific techniques and in which application scenario.

Evaluating Swirl and Tumble Motion

In the flow within a cylinder, we can distinguish between two types of motion: swirl flow commonly found in diesel engines and tumble flow commonly found in gas engines. In both cases, rotational motion occurs about an axis, though the position of the respective axis is different. In the case of swirl flow, the axis is more or less coincident with the cylinder axis, as shown in Figure 5.1 on page 63. In the

[4] For supplementary images and MPEG animations, please visit:
`http://www.VRVis.at/ar3/pr2/geometricApproach/`

case of tumble (Figure 7.1 on page 83), the rotation axis is perpendicular to the cylinder axis and more complex, thus making tumble flow more difficult to control than swirl flow.

In order to generate swirl or tumble motion, fluid enters the combustion chamber from the intake ports. Later on in the engine cycle, the kinetic energy associated with this motion is used to generate turbulence for mixing of fresh oxygen with evaporated fuel. The more turbulence generated, the better the mixture of air and fuel, and thus the more stable the combustion itself. By stable we mean achieving the same conditions for each engine cycle. Ideally, enough turbulent mixing is generated such that 100% of the fuel is burned. The swirl or tumble motion should be maximized to maximize turbulence. From the point of view of the mechanical engineers designing the intake ports, the ideal flow pattern leads to beneficial conditions including: improved mixture preparation, a higher EGR (Exhaust Gas Ratio) which means a decrease in fuel consumption, and lower emissions. However, too much swirl (or tumble) can displace the flame used to ignite the fuel, cause irregular flame propagation, or result in less fuel combustion. As such, a balance must be achieved between generating enough swirl or tumble flow and not displacing the flame used to ignite the flow. A controlled flow motion is used to get stable and reproducible conditions at each engine cycle.

Investigating Flow Patterns with Visualization

Central to our study are some routine questions that engineers may ask when investigating swirl and tumble flow: Can visualization provide insight into or verify the characteristic shape and behavior of the flow? What tools can help to visualize the swirl and tumble flow patterns? Where in the combustion chamber are the ideal swirl and tumble flow pattern *not* being realized? We investigated two typical flow patterns from CFD using three classes of flow visualization techniques commonly available in 2D, 2.5D, and 3D[5]. By 2.5D we mean surfaces through 3D space.

Figures 7.9 on page 94 and 7.7 on page 90 show some of the results of this investigation. Figure 7.9 is a hybrid of multiple visualization approaches including 3D streamlines, isosurfacing, color-mapping, and texture-based flow visualization. The streamlines highlight the dominant characteristic structure of the flow inside the volume while the texturing applied to the isosurface points out some of the destructive areas of the flow at the top. Figure 7.9 helps verify that the overall behavior of the flow is characteristic of that of swirl motion. Figure 7.7 is another hybrid of approaches including slicing with color-mapping and texture-based flow visualization applied, a pressure isosurface, and 3D streamlines seeded from two seeding planes. Figure 7.7 shows that in this case the tumble motion axis is off-center, pointing down and to the left rather than straight out towards the reader in the ideal case.

Our side-by-side comparison of each flow visualization category illustrates that each has its respective advantages and disadvantages. Direct flow visualization techniques are intuitive, easy to implement, common, and insightful. The direct flow visualization techniques are useful in highlighting extremal CFD simulation data values on surfaces. However, direct approaches may not communicate flow evolution very clearly and are often more difficult to apply in 3D. And important features can be missed if the sampling rate for a glyph-based representation is not high enough.

Geometric methods are also intuitive, provide insight, and can be applied to 2D, 2.5D, and 3D vector data. They also sometimes indicate flow direction including the upstream and downstream direction of the flow. Geometric techniques are useful for gaining insight into the location of the axis of rotation for both the swirl and tumble flow patterns while texture-based techniques provided useful enhancements to these results. However, the drawback with these approaches is generally that of placement. Important features may be overlooked because they do not provide complete coverage of the flow domain.

[5]For supplementary material including high resolution images and MPEG animations, please visit:
`http://www.VRVis.at/ar3/pr2/swirl-tumble/`

Texture-based approaches share advantages with both direct and geometric techniques by providing complete coverage and showing the direction of the flow everywhere. However, they are difficult to apply in 3D. The geometric and texture-based techniques applied to surfaces were very good at pointing out where the ideal swirl flow pattern was not being met in the CFD simulation model. The geometric techniques applied to slices and surfaces were also suitable for highlighting specific topological features of the flow whereas the texture-based approaches were very helpful by insuring that these topological features were not initially overlooked. Figure 7.8 on page 93 summarizes some of the trade-offs that are made when visualizing swirl and tumble motion. For example, The more dense the visualization, generally the more difficult is to perceive the result. Thus a trade-off is often made between these two factors. Also a trade-off may is made between spatial coherence and spatial dimensionality because results in 3D often contain many overlapping parts. As a result of these trade-offs, the flexible combinations of approaches offered by our system are good alternatives.

9.6 Discussion

The larger the data sets from CFD simulation become, the more useful scientific visualization is in order to gain insight into those results. In addition, no single “one-size-fits-all” approach exists, hence engineers require a range of tools in order to carry out their analysis, exploration, and presentation. We have presented a selection of recent advances in flow visualization. Our presentation draws upon flow visualization techniques being classified into four main categories. These recent advances have been applied to real-world applications in the field of CFD including the investigation and visualization of patterns of flow motion specific to automotive engineering. We have also discussed the benefits of these techniques and how engineers gain valuable insight into their CFD simulation results using this recently developed selection of tools.

Conclusions

"You've got to be careful if you don't know where you're going, 'cause you might not get there."

– Yogi Berra[6] (1925–)

From a researcher's point of view, the evolution of texture-based flow visualization techniques, in both 2D and 2.5D has taken big steps in the time frame from 2001-2004. Since the introduction of Spot Noise [163] and LIC [17], performance times for 2D texture-based methods have accelerated fairly steadily as evident from the large volume of research literature in this domain [82]. However, for the case of unsteady flow on surfaces, research had remained fairly static. To our knowledge, Shen and Kao were the first and last to address this challenge [139, 140] in 1998. The next significant progress addressing this problem came in 2003 [83, 165] -five years later. Similar is the case of 3D steady flow [153]. It is interesting to note the challenges that higher spatial and temporal dimensions represent and may always represent. The case of 3D, unsteady flow visualization still remains elusive.

From a practical point of view, it is indeed possible[7] to write a PhD and work at a private corporation full-time simultaneously, even while contributing published literature. This is by no means an obvious conclusion because of the conflicting interests represented by those that (1) sponsor the required money and (2) whose goal it is to publish research contributions. The semantics of *contribution* are relative. From an industry point of view, contribution means writing software, from a research scientists point of view, a contribution consists of published scientific literature. In fact, both are valuable contributions, although software contributions (i.e. source code) tend to lack in the research community and sound documentation tends to lack in the commercial software community.

Closely related is the subjectivity of what makes an effective visualization. In other words, visualization is also relative. Some members of the scientific visualization community voice skepticism concerning the results presented in field of information visualization (and vice-versa). Some members of the commercial software community express skepticism towards research results presented in the scientific visualization communities. And to complete the circle, the effectiveness of visualization presented by the commercial software industry is often criticized by members of the research community. Since there will always be criticism, the most important thing is that the criticism remain constructive.

One of the stronger criticisms coming from the commercial software industry is the research performed using programmable graphics hardware. During the time frame of this thesis (2001–2004), the amount of research literature involving GPU programming has expanded more-or-less exponentially. However, this leaves many from private industry raising the question: *How are we going to use this?* Since the work is

[6] Full Name: Lawrence Peter (Yogi) Berra, US baseball player, coach, and manager, catcher for New York Yankees 1946–1963, coach of New York Mets 1965–1972, New York Yankees 1975–1984, Houston Astros 1986–1989

[7] "possible" not to be confused with "easy"

the result of a collaboration between industry and research, all work presented here is platform independent. In industry, the customers have a strong influence on the software development requirements since they are a source of funding.

In the future, a greater understanding of the interests of commercial industry from the research community is the key to a successful collaboration between these two communities. The reverse is also true. A good place to start is by debunking the myth that the interests of industry and research do not overlap. In fact the overlap in goals can be very large.

Bibliography

[1] D. Arrowsmith and C. Place. *An Introduction to Dynamical Systems*. Cambridge University Press, 1990.

[2] G. V. Bancroft, F. J. Merritt, T. C. Plessel P. G. Kelaita, R. K. McCabe, and A. Globus. FAST: A Multiprocessed Environment for Visualization of Computational Fluid Dynamics. In *Proceedings of IEEE Visualization 90*, pages 14–27, 1990.

[3] H. Battke, D. Stalling, and H.C. Hege. Fast Line Integral Convolution for Arbitrary Surfaces in 3D. In *Visualization and Mathematics*, pages 181–195. Springer-Verlag, 1997.

[4] B. G. Becker, D. A. Lane, and N. L. Max. Unsteady Flow Volumes. In *Proceedings IEEE Visualization '95*, 1995.

[5] J. Becker and M. Rumpf. Visualization of Time-Dependent Velocity Fields by Texture Transport. In *Visualization in Scientific Computing '98*, Eurographics, pages 91–102, 1998.

[6] B. Beier and M. W. Vaughan. The bull's-eye: a framework for web application user interface design guidelines. In *Proceedings of ACM CHI 2003 Conference on Human Factors in Computing Systems*, volume 1 of *Web usability*, pages 489–496, 2003.

[7] S. Berger and M. E. Gröller. Color-Table Animation of Fast Oriented Line Intgral Convolution for Vector Field Visualization. In *WSCG 2000 Conference Proceedings*, pages 4–11, 2000.

[8] J. F. Blinn. Jim Blinn's Corner: Dirty Pixels. *IEEE Computer Graphics and Applications*, 9(4):100–105, July 1989.

[9] J. F. Blinn. Jim Blinn's Corner: Return of the Jaggy. *IEEE Computer Graphics and Applications*, 9(2):82–89, March 1989.

[10] U. Bordoloi and H. W. Shen. Hardware Accelerated Interactive Vector Field Visualization: A level of detail approach. In *Eurographics 2002 Proceedings*, volume 21(3) of *Computer Graphics Forum*, pages 605–614, 2002.

[11] E. Boring and A. Pang. Directional Flow Visualization of Vector Fields. In *Proceedings IEEE Visualization '95*, pages 389–392. IEEE, 1996.

[12] M. Botsch and L. P. Kobbelt. Resampling Feature and Blend Regions in Polygonal Meshes for Surface Anti-Aliasing. In *Eurographics 2001 Proceedings*, volume 20(3) of *Computer Graphics Forum*, pages 402–410. Blackwell Publishing, 2001.

[13] M. Brill, H. Hagen, H.-C. Rodrian, W. Djatschin, and S. V. Klimenko. Streamball Techniques for Flow Visualization. In *Proceedings IEEE Visualization '94*, pages 225–231. IEEE Computer Society, October 1994.

[14] S. Bryson and C. Levit. The Virtual Wind Tunnel. *IEEE Computer Graphics and Applications*, 12(4):25–34, July 1992.

[15] D. Bürkle, T. Preußer, and M. Rumpf. Transport and Anisotropic Diffusion in Time-Dependent Flow Visualization. In *Proceedings IEEE Visualization '01*, pages 61–67. IEEE Computer Society, 2001.

[16] B. Cabral and C. Leedom. Highly Parallel Vector Visualization Using Line Integral Convolution. In *Proceedings of the 27th Conference on Parallel Processing for Scientific Computing*, pages 802–807. SIAM Press, 1995.

[17] B. Cabral and L. C. Leedom. Imaging Vector Fields Using Line Integral Convolution. In *Poceedings of ACM SIGGRAPH 1993*, Annual Conference Series, pages 263–272. ACM Press / ACM SIGGRAPH, 1993.

[18] W. Cai and P. A. Heng. Principal Stream Surfaces. In *Proceedings IEEE Visualization '97*, pages 75–80. IEEE Computer Society, October 19–24 1997.

[19] N. A. Carr and J. C. Hart. Meshed Atlases for Real-Time Procedural Solid Texturing. *ACM Transactions on Graphics*, 21(2):106–131, April 2002.

[20] K. Chorianopoulos, G. Lekakos, and D. Spinellis. Intelligent User Interfaces in the Living Room: Usability Design for Personalized Television Applications. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces (IUI 03)*, pages 230–232. ACM Press, January 12–15 2003.

[21] J. Clyne and J. Dennis. Interactive Direct Volume Rendering of Time-Varying Data. In *Data Visualization '99*, Eurographics, pages 109–120. Springer-Verlag Wien, May 1999.

[22] S. D. Conte and C. de Boor. *Elementary Numerical Analysis*. McGraw-Hill, New York, 1980.

[23] A. Cooper and R. M. Reimann. *About Face 2.0: The Essentials of User Interface Design*. John Wiley & Sons, 2003.

[24] R. Crawfis, N. Max, B. Becker, and B. Cabral. Volume rendering of 3D scalar and vector fields at LLNL. In *Proceedings, Supercomputing '93: Portland, Oregon, November 15–19, 1993*, pages 570–576, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1993. IEEE Computer Society.

[25] R. A. Crawfis and N. Max. Texture Splats for 3D Scalar and Vector Field Visualization. In *Proceedings IEEE Visualization '93*, pages 261–267. IEEE Computer Society, October 1993.

[26] W. de Leeuw and R. van Liere. Visualization of Global Flow Structures Using Multiple Levels of Topology. In *Data Visualization '99*, Eurographics, pages 45–52. Springer-Verlag, May 1999.

[27] W. de Leeuw and R. van Liere. Multi-level Topology for Flow Visualization. *Computers and Graphics*, 24(3):325–331, June 2000.

[28] W. C. de Leeuw. Divide and Conquer Spot Noise. In *Proceedings of Supercomputing'97 (CD-ROM)*. ACM SIGARCH and IEEE, November 1997.

[29] W. C. de Leeuw, H.G. Pagendarm, F. H. Post, and B. Waltzer. Visual Simulation of Experimental Oil-Flow Visualization by Spot Noise from Numerical Flow Simulation. In *Visualization in Scientific Computing '95*, pages 135–148. Springer-Verlag, May 1995.

[30] W. C. de Leeuw, F. H. Post, and R. W. Vaatstra. Visualization of Turbulent Flow by Spot Noise. In *Virtual Environments and Scientific Visualization '96*, pages 286–295. Springer-Verlag, April 1996.

[31] W. C. de Leeuw and R. van Liere. Spotting Structure in Complex Time Dependent Flow. In H. Hagen, G. M. Nielson, and F. H. Post, editors, *Scientific Visualization*, pages 9–13. IEEE, June 1997. Dagstuhl Seminar 9724.

[32] W. C. de Leeuw and R. van Liere. Comparing LIC and Spot Noise,. In *Proceedings IEEE Visualization '98*, pages 359–366. IEEE Computer Society, 1998.

[33] W.C. de Leeuw and J.J. van Wijk. Enhanced Spot Noise for Vector Field Visualization. In *Proceedings IEEE Visualization '95*, pages 233–239. IEEE Computer Society, October 1995.

[34] U. Diewald, T. Preußer, and M. Rumpf. Anisotropic Diffusion in Vector Field Visualization on Euclidean Domains and Surfaces. In *IEEE Transactions on Visualization and Computer Graphics*, volume 6 (2), pages 139–149. IEEE Computer Society, 2000.

[35] H. Doleisch, M. Gasser, and H. Hauser. Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data. In *Proceedings of the 5th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003)*, pages 239–248, May 2003.

[36] H. Doleisch, M. Mayer, M. Gasser, R. Wanker, and H. Hauser. Case Study: Visual Analysis of Complex, Time-Dependent Simulation Results of a Diesel Exhaust System. In *Proceedings of the 6th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2004)*, pages 91–96, May 2004.

[37] D. S. Ebert and P. Rheingans. Volume Illustration: Nonphotorealistic Rendering of Volume Models. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(3), pages 253–264. IEEE Computer Society, 2001.

[38] D. S. Ebert and C. D. Shaw. Minimally Immersive Flow Visualization. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(4), pages 343–350. IEEE Computer Society, 2001.

[39] D. S. Ebert, R. Yagel, J. Scott, and Y. Kurzion. Volume Rendering Methods for Computational Fluid Dynamics Visualization. In *Proceedings IEEE Visualization '94*, pages 232–239. IEEE Computer Society, October 1994.

[40] A. Epstein and A. Beu. Design of a Graphical User Interface for Process Control Based on the Example of a Paper Recycling Plant. *International Journal of Human-Computer Interaction*, 14(3/4):387–400, 2002.

[41] G. Erlebacher, B. Jobard, and D. Weiskopf. Flow Textures. In C. R. Johnson and C. D. Hansen, editors, *The Visualization Handbook*. Academic Press, Oct 2003.

[42] A. J. Fenlon and T. David. An Integrated Visualization and Design Toolkit for Flexible Prosthetic Heart Valves. In *Proceedings IEEE Visualization 2000*, pages 453–456. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[43] L. K. Forssell. Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution. In *Proceedings IEEE Visualization '94*, pages 240–247. IEEE Computer Society, October 1994.

[44] L. K. Forssell and S. D. Cohen. Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, June 1995.

[45] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Object Technology Series. Addison-Wesley, third edition, September 2003.

[46] T. Frühauf. Raycasting vector fields. In *Proceedings IEEE Visualization '96*, pages 115–120. IEEE, October 27–November 1 1996.

[47] A. L. Fuhrmann and M. E. Gröller. Real-Time Techniques for 3D Flow Visualization. In *Proceedings IEEE Visualization '98*, pages 305–312. IEEE, 1998.

[48] T. Glau. Exploring instationary fluid flows by interactive volume movies. In *Data Visualization '99*, Eurographics, pages 277–283. Springer-Verlag Wien, May 1999.

[49] M. Hadwiger, C. Berger, and H. Hauser. High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware. In *Proceedings IEEE Visualization '03*, pages 301–308. IEEE Computer Society, 2003.

[50] M. Hadwiger, T. Theußl, H. Hauser, and E. Gröller. Hardware-Accelerated High-Quality Reconstruction on PC Hardware. In *Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV-01)*, pages 105–112, November 21–23 2001.

[51] H.C. Hege and D. Stalling. Fast LIC with Piecewise Polynomial Filter Kernels. In *Mathematical Visualization*, pages 295–314. Springer Verlag, 1998.

[52] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of Pixel Textures in Visualization and Realistic Image Synthesis. In *ACM Symposium on Interactive 3D Graphics*, pages 127–134, 1999.

[53] J. L. Helman and L. Hesselink. Representation and Display of Vector Field Topology in Fluid Flow Data Sets. *IEEE Computer*, 22(8):27–36, August 1989.

[54] J. L. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May 1991.

[55] L. Hesselink, F. H. Post, and J.J. van Wijk. Research Issues in Vector and Tensor Field Visualization. *IEEE Computer Graphics and Applications*, 14(2):76–79, March 1994.

[56] W. Hibbard. Connecting People to Computations and People to People. *Computer Graphics*, 32(3):10–12, 1998.

[57] W. Hibbard and D. Santek. Interactivity is the Key. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, pages 39–43, May 1989.

[58] J. P. M. Hultquist. Interactive Numerical Flow Visualization Using Stream Surfaces. *Computing Systems in Engineering*, 1(2-4):349–353, 1990.

[59] J. P. M. Hultquist. Constructing Stream Surfaces in Steady 3D Vector Fields. In *Proceedings IEEE Visualization '92*, pages 171–178. IEEE Computer Society, 1992.

[60] V. Interrante and C. Grosch. Strategies for Effectively Visualizing 3D Flow with Volume LIC. In *Proceedings IEEE Visualization '97*, pages 421–424, 1997.

[61] V. Interrante and C. Grosch. Visualizing 3D Flow. *IEEE Computer Graphics & Applications*, 18(4):49–53, 1998.

[62] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-Accelerated Texture Advection. In *Proceedings IEEE Visualization 2000*, pages 155–162. IEEE Computer Society, 2000.

[63] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian Advection for Unsteady Flow Visualization. In *Proceedings IEEE Visualization '01*. IEEE, October 2001.

[64] B. Jobard, G. Erlebacher, and Y. Hussaini. Lagrangian-Eulerian Advection of Noise and Dye Textures for Unsteady Flow Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.

[65] B. Jobard and W. Lefer. Creating Evenly–Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97*, volume 7. Eurographics, Springer-Verlag, 1997.

[66] B. Jobard and W. Lefer. The Motion Map: Efficient Computation of Steady Flow Animations. In *Proceedings IEEE Visualization '97*, pages 323–328. IEEE Computer Society, October 19–24 1997.

[67] B. Jobard and W. Lefer. Unsteady Flow Visualization by Animating Evenly-Spaced Streamlines. In M. Gross and F. R. A. Hopgood, editors, *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), 2000.

[68] B. Jobard and W. Lefer. Multiresolution Flow Visualization. In *WSCG 2001 Conference Proceedings*, Plzen, Czech Republic, February 2001.

[69] D. Kao, B. Zhang, K. Kim, and A. Pang. 3D Flow Visualization Using Texture Advection. In *International Conference on Computer Graphics and Imaging '01*, August 2001.

[70] D. N. Kenwright and D. A. Lane. Interactive Time-Dependent Particle Tracing Using Tetrahedral Decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):120–129, June 1996.

[71] L. Khouas, C. Odet, and D. Friboulet. 2D Vector Field Visualization Using Furlike Texture. In *Joint Eurographics-IEEE TVCG Symposium on Visualization (VisSym '99)*, Eurographics, pages 35–44. Springer-Verlag Vienna, May 1999.

[72] R. M. Kirby, H. Marmanis, and D. H. Laidlaw. Visualizing Multivalued Data from 2D Incompressible Flows Using Concepts from Painting. In *Proceedings IEEE Visualization '99*, pages 333–340. ACM Press, October 25–29 1999.

[73] M.H. Kiu and D. C. Banks. Multi-frequency Noise for LIC. In *Proceedings IEEE Visualization '96*, pages 121–126. IEEE, October 27–November 1 1996.

[74] R. V. Klassen and S. J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proceedings IEEE Visualization '91*, pages 148–153, 1991.

[75] D. H. Laidlaw, R. M. Kirby, J. S. Davidson, T. S. Miller, M. da Silva, W. H. Warren, and M. Tarr. Quantitative Comparative Evaluation of 2D Vector Field Visualization Methods. In *Proceedings IEEE Visualization 2001*, pages 143–150. IEEE Computer Society, October 2001.

[76] D. A. Lane. *Scientific Visualization of Large-Scale Unsteady Fluid Flows*, chapter 5, pages 125–145. Scientific Visualization: Overviews, Methodologies, and Techniques. IEEE Computer Science Press, 1997.

[77] R. S. Laramee. Interactive 3D Flow Visualization Using a Streamrunner. In *CHI 2002, Conference on Human Factors in Computing Systems, Extended Abstracts*, pages 804–805. ACM SIGCHI, ACM Press, April 20–25 2002.

[78] R. S. Laramee. FIRST: A Flexible and Interactive Resampling Tool for CFD Simulation Data. *Computers & Graphics*, 27(6):905–916, 2003.

[79] R. S. Laramee and R. D. Bergeron. An Isosurface Continuity Algorithm for Super Adaptive Resolution Data. In *Advances in Modelling, Animation, and Rendering: Computer Graphics International (CGI 2002)*, pages 215–237. Computer Graphics Society, Springer, July 1-5 2002.

[80] R. S. Laramee and H. Hauser. Geometric Flow Visualization Techniques for CFD Simulation Data. In *VRVis Research Center Technical Report TR-VRVis-2004-029*, Vienna, Austria, Aug 2004.

[81] R. S. Laramee and H. Hauser. Interactive 3D Flow Visualization Using Textures and Geometric Primitives. In *NAFEMS World Congress Conference Proceedings*. NAFEMS–The International Association for the Engineering Analysis Community, May 17–20 2005. forthcoming.

[82] R. S. Laramee, H. Hauser, H. Doleisch, F. H. Post, B. Vrolijk, and D. Weiskopf. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum*, 23(2):203–221, June 2004.

[83] R. S. Laramee, B. Jobard, and H. Hauser. Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proceedings IEEE Visualization '03*, pages 131–138. IEEE Computer Society, 2003.

[84] R. S. Laramee, J. Schneider, and H. Hauser. Texture-Based Flow Visualization on Isosurfaces from Computational Fluid Dynamics. In *Data Visualization, The Joint Eurographics-IEEE TVCG Symposium on Visualization (VisSym '04)*, pages 85–90,342. Eurographics Association, 2004.

[85] R. S. Laramee, J. J. van Wijk, B. Jobard, and H. Hauser. ISA and IBFVS: Image Space Based Visualization of Flow on Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):637–648, November 2004.

[86] R. S. Laramee, D. Weiskopf, J. Schneider, and H. Hauser. Investigating Swirl and Tumble Flow with a Comparison of Visuaization Techniques. In *Proceedings IEEE Visualization '04*, pages 51–58. IEEE Computer Society, 2004.

[87] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, pages 362–371, 2002.

[88] G. S. Li, U. Bordoloi, and H. W. Shen. Chameleon: An Interactive Texture-based Framework for Visualizing Three-dimensional Vector Fields. In *Proceedings IEEE Visualization '03*, pages 241–248. IEEE Computer Society, 2003.

[89] Z. P. Liu and R. J. Moorhead, II. AUFLIC: An Accelerated Algorithm for Unsteady Flow Line Integral Convolution. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym '02)*, pages 43–52, 2002.

[90] S. K. Lodha, A. Pang, R. E. Sheehan, and C. M. Wittenbrink. UFLOW: Visualizing Uncertainty in Fluid Flow. In *Proceedings IEEE Visualization '96*, pages 249–254, October 27–November 1 1996.

[91] H. Löffelmann and M. E. Gröller. Enhancing the Visualization of Characteristic Structures in Dynamical Systems. In *Proceedings of the 9th Eurographics Workshop on Visualization in Scientific Computing*, Eurographics, pages 35–46. Springer-Verlag, 1998.

[92] H. Löffelmann, A. König, and E. Gröller. Fast Visualization of 2D Dynamical Systems by the Use of Virtual Ink Droplets. In *13th Spring Conference on Computer Graphics*, pages 111–118. Comenius University, Bratislava, Slovakia, June 1997.

[93] H. Löffelmann, T. Kučera, and E. Gröller. Visualizing poincaré Maps Together with the Underlying Flow. In *Mathematical Visualization*, pages 315–328. Springer Verlag, 1998.

[94] H. Löffelmann, L. Mroz, and E. Gröller. Hierarchical Streamarrows for the Visualization of Dynamical Systems. In *Visualization in Scientific Computing '97*, Eurographics, pages 155–164. Springer-Verlag, 1997.

[95] H. Löffelmann, L. Mroz, E. Gröller, and W. Purgathofer. Stream Arrows: Enhancing the Use of Streamsurfaces for the Visualization of Dynamical Systems. *The Visual Computer*, 13:359–369, 1997.

[96] H. Löffelmann, Z. Szalavàri, and E. Gröller. Local Analysis of Dynamical Systems – Concepts and Interpretation. In *WSCG 1996 Conference Proceedings*, pages 170–180, February 1996.

[97] W. E. Lorensen and H. E. Cline. Marching Cubes: a High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, pages 163–170. ACM, July 27–31 1987.

[98] M. M. Loughlin and J. F. Hughes. An Annotation System for 3D Fluid Flow Visualization. In *Proceedings IEEE Visualization '94*, pages 273–280. IEEE Computer Society, October 1994.

[99] X. Mao, Y. Hatanaka, H. Higashida, and A. Imamiya. Image-Guided Streamline Placement on Curvilinear Grid Surfaces. In *Proceedings IEEE Visualization '98*, pages 135–142. IEEE, 1998.

[100] X. Mao, L. Hong, A. Kaufman, N. Fujita, and M. Kikukawa. Multi-Granularity Noise for Curvilinear Grid LIC. In *Graphics Interface*, pages 193–200, June 1998.

[101] X. Mao, M. Kikukawa, N. Fujita, and A. Imamiya. Line Integral Convolution for 3D Surfaces. In *Visualization in Scientific Computing '97. Proceedings of the Eurographics Workshop*, pages 57–70. Eurographics, 1997.

[102] N. Max and B. Becker. Flow Visualization Using Moving Textures. In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, September 1995.

[103] N. Max and B. Becker. Flow Visualization Using Moving Textures. In *Data Visualization Techniques*, pages 99–105, 1999.

[104] N. Max, B. Becker, and R. Crawfis. Flow Volumes for Interactive Vector Field Visualization. In *Proceedings IEEE Visualization '93*, pages 19–24. IEEE Computer Society, October 1993.

[105] N. Max, R. Crawfis, and D. Williams. Visualizing Wind Velocities by Advecting Cloud Textures. In *Proceedings IEEEVisualization '92*. IEEE Computer Society, 1992.

[106] P. Moran, C. Henze, D. Ellsworth, S. Bryson, and D. Kenwright. The Field Encapsulation Library (FEL). The Field Encapsulation Library (FEL) is a library for representing fields and the meshes that fields are based on, see http://www.nas.nasa.gov/Groups/VisTech/projects/fel/.

[107] J. Nielsen. Interface: Using Paper Prototypes in Home-Page Design. *IEEE Software*, 12(4):88–89, 97, July 1995.

[108] G. M. Nielson. Tools for Computing Tangent Curves for Linearly Varying Vector Fields over Tetrahedral Domains. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):360–372, October – December 1999. ISSN 1077-2626.

[109] G. M. Nielson, H. Hagen, and H. Müller. *Scientific Visualization: Overviews, Methodologies, and Techniques*. IEEE Computer Society, 1997.

[110] A. Okada and David L. Kao. Enhanced Line Integral Convolution with Flow Feature Detection. In *SPIE Vol. 3017 Visual Data Exploration and Analysis IV*, pages 206–217, February 1997.

[111] K. Ono, H. Matsumoto, and R. Himeno. Visualization of Thermal Flows in an Automotive Cabin with Volume Rendering Method. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym '01)*, pages 301–308. Springer-Verlag, May 28–30 2001.

[112] D. R. Peachey. Solid Texturing of Complex Surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, 19(3):279–286, 1985.

[113] C. Pickover and S. Tewksbury. *Frontiers of Scientific Visualization*. Wiley Interscience, 1993.

[114] F. H. Post and T. van Walsum. Fluid flow visualization. In *Focus on Scientific Visualization*, pages 1–40. Springer, 1993.

[115] F. H. Post and J.J. van Wijk. *Visual Representation of Vector Fields: Recent Developments and Research Directions*, chapter 23, pages 367–390. Springer, 1994. in: L. Rosenblum et al. (eds.), Scientific Visualization: Advances and Challenges.

[116] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. Feature Extraction and Visualization of Flow Fields. In *Eurographics 2002 State-of-the-Art Reports*, pages 69–100. The Eurographics Association, 2–6 September 2002.

[117] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The State of the Art in Flow Visualization: Feature Extraction and Tracking. *Computer Graphics Forum*, 22(4):775–792, Dec. 2003.

[118] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2 edition, 2002.

[119] T. Preußer and M. Rumpf. Anisotropic Nonlinear Diffusion in Flow Visualization. In *Proceedings IEEE Visualization '99*, pages 325–332. IEEE Computer Society, October 1999.

[120] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing. Principles, Algorithms, and Applications*. Printice Hall International, third edition, 1996.

[121] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping. In *Proceedings IEEE Visualization '99*, pages 233–240. IEEE Computer Society, 1999.

[122] P. J. Rhodes, R. D. Bergeron, and T. M. Sparr. A Data Model for Multiresolution Scientific Data Environments. In *NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization*, October 15–17 2000.

[123] P. J. Rhodes, R. D. Bergeron, and T. M. Sparr. A Data Model for Distributed Multiresolution Multisource Scientific Data. In G. Farin, H. Hagen, and B. Hamann, editors, *Hierarchical and Geometrical Methods in Scientific Visualization*, 2002.

[124] P. J. Rhodes, R. S. Laramee, R. D. Bergeron, and T. M. Sparr. Uncertainty Visualization Methods in Isosurface Rendering. In M. Chover, H. Hagen, and D. Tost, editors, *Eurographics 2003, Short Papers*, pages 83–88. The Eurographics Association, September 1-5 2003.

[125] C. Rocchini, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, and R. Scopigno. Marching Intersections: An Efficient Resampling Algorithm for Surface Management. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI 01)*, pages 296–305. IEEE Computer Society, May 7–11 2001.

[126] S. Röttger, M. Kraus, and T. Ertl. Hardware-Accelerated Volume and Isosurface Rendering Based on Cell-Projection. In *Proceedings IEEE Visualization 2000*, pages 109–116. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[127] I. A. Sadarjoen, A. J. de Boer, F. H. Post, and A. E. Mynett. Particle Tracing in σ-Transformed Grids Using Tetrahedral 6-Decomposition. In *Visualization in Scientific Computing '98*, Eurographics, pages 71–80. Springer-Verlag Wien New York, 1998.

[128] A. Sanna, B. Montrucchio, and R. Arinaz. Visualizing Unsteady Flows by Adaptive Streaklines. In *WSCG 2000 Conference Proceedings*, pages 84–91, 2000.

[129] A. Sanna, B. Montrucchio, and P. Montuschi. A Survey on Visualization of Vector Fields by Texture-Based Methods. *Recent Research Developments in Pattern Recognition*, 1(1):13–27, 2000.

[130] A. Sanna, B. Montrucchio, P. Montuschi, and A. Sparavigna. Visualizing Vector Fields: The Thick Oriented Stream-line Algorithm (TOSL). *Computers and Graphics*, 25(5):847–855, October 2001.

[131] A. Sanna, C. Zunino, B. Montucchio, and P. Montuschi. Adding a Scalar Value to Texture-Based Vector Field Representations by Local Contrast Analysis. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym '02)*, pages 35–41, 2002.

[132] G. Scheuermann, H. Burbach, and H. Hagen. Visualizing Planar Vector Fields with Normal Component Using Line Integral Convolution. In *Proceedings IEEE Visualization '99*, pages 255–262. IEEE Computer Society, 1999.

[133] B. Schneiderman. *Designing the User Interface: Strategies for Effective Computer Interaction*. Addison-Wesley, 2nd edition, 1992.

[134] W. Schroeder, C. R. Volpe, and W. E. Lorensen. The Stream Polygon: A Technique for 3D Vector Field Visualization. In *Proceedings IEEE Visualization '91*, pages 126–132, 1991.

[135] W. J. Schroeder, K. M. Martin, and W. E. Lorensen. *The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics*. Prentice-Hall, 2nd edition, 1998.

[136] W. J. Schroeder, K. M. Martin, and W. E. Lorensen. *The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics*. Kitware, Inc., 3rd edition, 2003.

[137] M. Schulz, F. Reck, W. Bartelheimer, and T. Ertl. Interactive Visualization of Fluid Dynamics Simulations in Locally Refined Cartesian Grids. In *Proceedings IEEE Visualization '99*, pages 413–416, 1999.

[138] H. W. Shen, C. R. Johnson, and K. L. Ma. Visualizing Vector Fields Using Line Integral Convolution and Dye Advection. In *1996 Volume Visualization Symposium*, pages 63–70. IEEE, October 1996.

[139] H. W. Shen and D. L. Kao. UFLIC: A Line Integral Convolution Algorithm for Visualizing Unsteady Flows. In *Proceedings IEEE Visualization '97*, pages 317–323. IEEE Computer Society, 1997.

[140] H.W. Shen and D. L. Kao. A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, April – June 1998.

[141] P. Shirley and A. Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 63–70, November 1990.

[142] D. Silver, F. Post, and I. Sadarjoen. *Flow Visualization*, volume 7 of *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 640–652. John Wiley & Sons, 1999.

[143] D. Stalling. LIC on Surfaces. In *Texture Synthesis with Line Integral Convolution*, pages 51–64. ACM SIGGRAPH 97, International Conference on Computer Graphics and Interactive Techniques, 1997.

[144] D. Stalling and H. Hege. Fast and Resolution Independent Line Integral Convolution. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 249–256. ACM SIGGRAPH, ACM Press / ACM SIGGRAPH, 1995.

[145] A. Sundquist. Dynamic Line Iintegral Convolution for Visualizing Streamline Evolution. In *IEEE Transactions on Visualization and Computer Graphics*, volume 9(3), pages 273–282, 2003.

[146] J. E. Swan, M. Lanzagorta, D. Maxwell, E. Kou, J. Uhlmann, W. Anderson, H.J. Shyu, and W. Smith. A Computational Steering System for Studying Microwave Interactions with Missile Bodies. In *Proceedings IEEE Visualization 2000*, pages 441–444. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[147] C. K. Tang and G. G. Medioni. Extremal Feature Extraction From 3D Vector and Noisy Scalar Fields. In *Proceedings IEEE Visualization '98*, pages 95–102. IEEE, 1998.

[148] F. Taponecco and M. Alexa. Vector Field Visualization using Markov Random Field Texture Synthesis. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym '03)*, pages 195–202. Springer-Verlag, may 2003.

[149] C. Teitzel and T. Ertl. New Approaches for Particle Tracing on Sparse Grids. In *Data Visualization '99*, Eurographics, pages 73–84. Springer-Verlag, May 1999.

[150] C. Teitzel, R. Grosso, and T. Ertl. Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes. In *Visualization in Scientific Computing '97*, Eurographics, pages 31–42. Springer-Verlag Wien New York, 1997.

[151] C. Teitzel, R. Grosso, and T. Ertl. Line Integral Convolution on Triangulated Surfaces. In *WSCG 1997 Conference Proceedings*, pages 572–581, 1997.

[152] C. Teitzel, R. Grosso, and T. Ertl. Particle Tracing on Sparse Grids. In *Visualization in Scientific Computing '98*, Eurographics, pages 81–90. Springer-Verlag Wien New York, 1998.

[153] A. Telea and J. J. van Wijk. 3D IBFV: Hardware-Accelerated 3D Flow Visualization. In *Proceedings IEEE Visualization '03*, pages 233–240. IEEE Computer Society, 2003.

[154] A. Telea and J.J. van Wijk. Simplified Representation of Vector Fields. In *Proceedings IEEE Visualization '99*, pages 35–42. IEEE Computer Society, 1999.

[155] R. J. Torres. *Practitioners Handbook for User Interface Design and Development*. Prentice Hall, 2002.

[156] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised Marching Tetrahedra: Improved Isosurface Extraction. *Computers and Graphics*, 23(4):583–598, August 1999.

[157] L. A. Treinish. Multi-Resolution Visualization Techniques for Nested Weather Models. In *Proceedings IEEE Visualization 2000*, pages 513–516. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[158] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[159] G. Turk and D. Banks. Image-Guided Streamline Placement. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 453–460. ACM SIGGRAPH, Addison Wesley, August 1996.

[160] S. K. Ueng, C. Sikorski, and K. L. Ma. Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):100–110, June 1996.

[161] T. Urness, V. Interrante, I. Marusic, E. Longmire, and B. Ganapathisubramani. Effectively Visualizing Multi-Valued Flow Data using Color and Texture. In *Proceedings IEEE Visualization '03*, pages 115–122. IEEE Computer Society, 2003.

[162] M. van Dyke. *An Album of Fluid Motion*. The Parabolic Press, 1982.

[163] J. J. van Wijk. Spot noise-Texture Synthesis for Data Visualization. In Thomas W. Sederberg, editor, *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, volume 25, pages 309–318. ACM, 1991.

[164] J. J. van Wijk. Image Based Flow Visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

[165] J. J. van Wijk. Image Based Flow Visualization for Curved Surfaces. In *Proceedings IEEE Visualization '03*, pages 123–130. IEEE Computer Society, 2003.

[166] J.J. van Wijk. Flow Visualization with Surface Particles. *IEEE Computer Graphics and Applications*, 13(4):18–24, July 1993.

[167] J.J. van Wijk. Implicit Stream Surfaces. In *Proceedings of the Visualization '93 Conference*, pages 245–252. IEEE Computer Society, October 1993.

[168] V. Verma, D. Kao, and A. Pang. PLIC: Bridging the Gap Between Streamlines and LIC. In *Proceedings IEEE Visualization '99*, pages 341–348, October 25–29 1999.

[169] V. Verma, D. Kao, and A. Pang. A Flow-guided Streamline Seeding Strategy. In *Proceedings IEEE Visualization 2000*, pages 163–170, 2000.

[170] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Addison-Wesley, February 1996.

[171] C. Ware and G. Franck. Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions. *ACM Transactions on Graphics*, 15(2):121–140, April 1996.

[172] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2000.

[173] G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, and K. I. Joy. Extraction of Crack-free Isosurfaces From Adaptive Mesh Refinement Data. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym-01)*, pages 25–34. Springer-Verlag, May 28–30 2001.

[174] R. Wegenkittl, E. Gröller, and W. Purgathofer. Visualizing the Dynamical Behavior of Wonderland. *IEEE Computer Graphics and Applications*, 17(6):71–79, November/December 1997.

[175] R. Wegenkittl and M. E. Gröller. Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet. In *Proceedings IEEE Visualization '97*, pages 309–316, October 19–24 1997.

[176] R. Wegenkittl, M. E. Gröller, and W. Purgathofer. Animating Flow Fields: Rendering of Oriented Line Integral Convolution. In *Computer Animation '97 Proceedings*, pages 15–21, June 1997.

[177] M. Weiler and T. Ertl. Hardware-Software-Balanced Resampling for the Interactive Visualization of Unstructured Grids. In *Proceedings IEEE Visualization 2001*, pages 199–206, 2001.

[178] D. Weiskopf, G. Erlebacher, and T. Ertl. A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields. In *Proceedings IEEE Visualization '03*, pages 107–114. IEEE Computer Society, 2003.

[179] D. Weiskopf, G. Erlebacher, M. Hopf, and T. Ertl. Hardware-Accelerated Lagrangian-Eulerian Texture Advection for 2D Flow Visualizations. In *Proceedings of the Vision Modeling and Visualization Conference 2002 (VMV-01)*, pages 439–446, November 21–23 2002.

[180] D. Weiskopf and T. Ertl. GPU-Based 3D Texture Advection for the Visualization of Unsteady Flow Fields. In *WSCG 2004 Conference Proceedings, Short Papers*, pages 259–266, February 2004.

[181] D. Weiskopf, M. Hopf, and T. Ertl. Hardware-Accelerated Visualization of Time-Varying 2D and 3D Vector Fields by Texture Advection via Programmable Per-Pixel Operations. In *Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV 01)*, pages 439–446, November 21–23 2001.

[182] R. Westermann. The Rendering of Unstructured Grids Revisited. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym-01)*, pages 65–74. Springer-Verlag, May 28–30 2001.

[183] R. Westermann, C. Johnson, and T. Ertl. Topology-preserving smoothing of vector fields. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(3), pages 222–229. IEEE Computer Society, 2001.

[184] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall, 1990.

[185] R. Yagel, D. M. Reed, A. Law, P. Shih, and N. Shareef. Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing. In *Proceedings 1996 Symposium on Volume Visualization*, pages 55–62, September 1996.

[186] M. Zöckler, D. Stalling, and H. Hege. Interactive Visualization of 3D-Vector Fields Using Illuminated Streamlines. In *Proceedings IEEE Visualization '96*, pages 107–113, October 1996.

[187] M. Zöckler, D. Stalling, and H.-C. Hege. Parallel Line Integral Convolution. *Parallel Computing*, 23(7):975–989, 1997.

Curriculum Vitae

Robert S. Laramee
born on 23 October 1973, in Boston, Massachusetts.

Education

- PhD candidate in Computer Science, Vienna University of Technology, Austria
- MSc in Computer Science, University of New Hampshire, Durham, NH, 2000
- BS in Physics with a minor in Computer Science, Cum Laude, University of Massachusetts, Amherst, MA, 1997
- Study of Physics and Computer Science, University of Hull, Kingston-Upon-Hull, England, 1995-96

Experience

2001–present
VRVis Research Center
Donau-City-Strasse 1
1220 Vienna, Austria
`http://www.VRVis.at`
Junior Researcher and Software Engineer

VRVis is a research center that attempts to form a transfer-of-knowledge bridge between universities and the commercial industry sector in Austria. In this specific case, the commercial sector is AVL (`www.avl.com`) and the university is Vienna University of Technology, Austria. A junior researcher is responsible for research and development in the area of scientific visualization. The focus of the research is in the area flow visualization and the software, written in C++, is developed jointly with AVL. All development is carried out directly within the commercial software environment of AVL in order to take advantage of any commercial benefits that may stem from the research. AVL's software is designed to visualize the results of computational fluid dynamics (CFD) simulations. Please see list of publications for related research resulting from the collaboration.

1998–2001
Department of Computer Science
University of New Hampshire
Durham, NH 03824, U.S.
`http://www.cs.unh.edu/`
Research Assistant

Responsible for research in the fields of computer graphics, scientific visualization, and human-computer interaction (HCI). The focus of the scientific visualization was in the area of multiresolution (MR) and

adaptive resolution (AR) visualization. We implemented an isosurface renderer that supports visualization of both MR and AR data. The software implementation of our isosurface renderer was written in Java 1.2/2.0 and used the VisAD, Java3D, and Mesa/OpenGL graphics libraries. The focus of the HCI research was on the visual problems associated with the use of monocular head mounted displays. We carried out a series of user studies that emphasize perceptual problems related to the use of this type of display. Please see list of publications for related research resulting from this work.

1997–2000
Department of Computer Science
University of New Hampshire
Durham, NH, 03824, U.S.
`http://www.cs.unh.edu/`
Computer Applications Lecturer/Instructor

Responsible for teaching a survey computer science course at UNH that provides a theoretical overview in the topics of operating systems, computer architecture, networking, computer graphics and databases. The course also introduced the features of the UNIX, Macintosh, and Windows operating systems as well as popular business office software including word processing, spreadsheet, and simple database applications. Approximately 475 students attended the lecture over the course of four years.

Professional Activities

- reviewer for the IEEE Visualization conference
- reviewer for IEEE Transactions on Visualization and Computer Graphics
- member of EUROGRAPHICS, the European Association for Computer Graphics. 2003-2004
- member, IEEE Computer Society

Publications

Robert S. Laramee
Please see the **Related Publications** or **Bibliography** sections for a list of publications. For a full, more up-to-date list, including supplementary images, animations, and related work, please visit:
`http://www.VRVis.at/ar3/pr2/laramee/`

Contact Information

Robert S. Laramee, MarienGasse 5, 8020 Graz, Austria,
`Laramee ‘‘at’’ VRVis.at`

Acknowledgements

"Be who you are and say what you feel, because those who mind don't matter and those who matter don't mind."

–Dr. Seuss[8] (1904–1991)

This thesis was the work of (myself and) many people from far away. While the content was written in Graz [9], Austria, many people have contributed to the work shown here, almost none of whom were in Graz, and each whose contribution was essential. In fact, this thesis was an exercise in remote collaboration. Therefore, we acknowledge the contributers, to whom we are very grateful, in the form of the table below (last updated on Friday 26 Nov 2004), identifying: (1) the contributer, (2) affiliation, (3) the distance, in kilometers, the contributer is from Graz [10], (4) the approximate number of emails received by the contributer during the course of the thesis (5) the role played by the contributer. The "strength" of the collaboration can be judged roughly by the number of emails received, although not included are the important tele-meetings that took place.

It is worthy to note that the no. of emails received column contains some error. In fact, the author's record of email dates back only to 10 December 2001. This is because in late November/early December of 2001, EDS (Electronic Data Systems), the information technology solutions provider for AVL-AST, decided to upgrade the author's workstation to Red Hat version 7.x. In doing so, all of the author's data was deleted from the workstation, including any locally stored email. Not only was the entire hard drive erased, but the upgrade failed due to a problem with the graphics card driver (for the *HP fx* graphics card). So the previous version of the Red Hat OS was re-installed. Interestingly, EDS decided to wait until the author was on one of his trips to Vienna to attempt the upgrade. The author was surprised to discover the state of the aforementioned workstation upon his return to Graz.

R Daniel Bergeron, Philip J. Rhodes, and Ted M. Sparr are with the Department of Computer Science, University of New Hampshire, Durham, NH 03824, although Phil Rhodes, as of August 2004, has moved to the Department of Computer Science at the University of Mississippi, Oxford, MS. Before joining VRVis, the author was in the Scientific Database Group (SDB) at UNH with Dan, Phil, and Ted. This was very good preparation for the work awaiting him in Austria. Helmut Doleisch, Markus Hadwiger, and Helwig Hauser are with the VRVis Research Center, Donau-City-Strasse 1, 1220 Vienna, Austria. Meister Eduard Gröller and Werner Purgathofer are at the Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria. Bruno Jobard is at the University of Pau, France. Although Bruno was at the Swiss Center for Scientific Computing (CSCS), in Manno Switzerland, approximately 520 kilometers away, when this thesis started. So he moved further away. Thomas Laramee

[8] US author and illustrator, wrote "The 500 Hats of Bartholomew Cubbins" 1938, "The Cat in the Hat" 1957", "How the Grinch Stole Christmas" 1957, "Green Eggs and Ham" 1970, "The Lorax" 1971

[9] The thesis was written at AVL AST, Alte Post Strasse 152, 8020 Graz, Austria, Cultural Capital of Europe, 2003

[10] computed by `http://www.GeoBytes.com/`

Name	Distance from Graz (km)	No. of emails received	Role
R. Daniel Bergeron	6,443	16	former adviser, coauthor
Helmut Doleisch	143	102	coauthor
Markus Hadwiger	143	35	technical support
Helwig Hauser	143	601	adviser, coauthor
Meister Eduard Gröller	143	20	spiritual presence
Bruno Jobard	1,306	155	coauthor
Thomas Laramee	8,729	≈60	web hosting
Frits H. Post	969	55	coauthor
Werner Purgathofer	143	70	adviser, string puller
Philip J. Rhodes	6,443	5	coauthor
Jürgen Schneider	0	7	coauthor
Ted M. Sparr	6,443	1	coauthor
Benjamin Vrolijk	969	26	coauthor
Colin Ware	6,443	13	coauthor
Daniel Weiskopf	504	147	coauthor
Jarke J. van Wijk	872	174	coauthor

Table 9.1: Thesis technical contributers.

has a web server at home, which the author used: 4207 Dayton Ave N., Seattle, WA 98103 US. Frits H. Post and Benjamin Vrolijk are with The Computer Graphics and CAD/CAM Group, Delft University of Technology, The Netherlands. Jürgen Schneider works for AVL AST ("Anstalt für Verbrennungskraftmaschinen" Advanced Simulation Technologies), Alte Post Strasse 152, 8020 Graz, Austria. Daniel Weiskopf is with The Institute of Visualization and Interactive Systems, University of Stuttgart, Germany. Jarke J. van Wijk is in The Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, The Netherlands. It is interesting to note the unexpected, powerful role that the Dutch (Benjamin, Jack, and Frits) played. Also worthy of mention are Kresimir Matkovic of VRVis for his support and Bernhard Koller in the Studien- und Prüfungsabteilung at the Vienna University of Technology. Bernie helped me through the university's admittance process, a laborious process that in fact took over six months. We've exchanged more than 26 emails just to get into the PhD program.

These people are very good to work with.

We thank all organizations who have contributed to this research including AVL, the Austrian national research program called K plus (`www.kplus.at`), and the VRVis Research Center (`www.VRVis.at`). Furthermore, the author thanks all the colleagues from the research community who reviewed our publications. CFD simulation data courtesy of AVL. We also thank EDS Austria for their technical support.

Appendix: Bridging the Gap Between Industry and Research

"Human felicity is produc'd not so much by great Pieces of good Fortune that seldom happen, as by little Advantages that occur every day."

–Benjamin Franklin[11] (1706–1790)

In this case, industry is represented by AVL in Graz and research is represented by Vienna University of Technology. The VRVis Research Center in Vienna represents the bridge between university and industry and collaborates closely with both. Since the thesis author's workstation was located in AVL AST Graz, and the thesis author's research adviser(s) was located in VRVis, Vienna, many trips were made, by train, between Graz and Vienna. These trips represented the gap between industry and research in more ways than one. We summarize, in tabular form (last updated on Friday 26 November 2004), the trips made between Vienna and Graz by the thesis author during the course of this PhD. Note that not all trips

2001				
August 02	August 16	August 30	September 6	September 27
October 11	November 8	November 29	December 6	December 20
2002				
January 10	January 24	February 14	March 7	April 4-5
April 18	May 16-17	June 13	June 27	August 8
September 26	October 10	October 24	November 14	December 12
2003				
January 21	January 30	February 13	March 13	March 27
May 8	May 22	June 26	July 24	August 21
September 18	October 16	November 13-14		
2004				
January 15	January 29	February 25	March 25	April 29
May 13	May 14	May 26	June 24	July 22
August 19	September 9	September 23	November 17	

Table 9.2: Round trip journeys between Graz and Vienna taken by the author during the course of this PhD. Note, dates with a dash represent overnight stays associated with invited VRVis Forum talks.

to Vienna are included, only business trips. Some private trips are missing. The total number of listed trips is 52. The distance between Graz and Vienna is 143 km (as the crow flies). Thus a distance, d of

[11] US author, diplomat, inventor, physicist, politician, and printer, published "Poor Richard's Almanack" 1732–1757, founder and 1^{st} president of American Philosophical Society 1769–1790

approximately:

$$d = 52\ trips \times 143\ km/stretch \times 2\ stretches/trip = 14,872\ km \tag{9.2}$$

was covered. The distance traveled by the train is slightly longer as it is not straight. Each round trip took approximately six hours. So the time, t, spent commuting was approximately:

$$t = 6\ hours/trip \times 52\ trips = 312\ hours \tag{9.3}$$

or about eight, forty hour work weeks (roughly two months).

Index

2.5D, 8, 9
  fast LIC, 18
  geometric flow visualization, 29
  IBFVS, 23
  ISA, 23
  LIC, 17, 20
  spot noise, 15
  streamline seeding, 29
  surface particles, 31
  swirl motion, 87
  tumble motion, 87
2D, 8
  contouring, 27
  geometric flow visualization, 28
  IBFV, 23
  LEA, 25
  LIC, 17
  moving textures, 23
  spot noise, 15
  streamlets, 28
  streamlines, 28
  streamline seeding, 28
  swirl motion, 84
  tumble motion, 85
  UFAC, 26
3D, 8
  dash tube, 31
  flow volume, 32
  geometric flow visualization, 29
  IBFV, 24
  isosurface, 28
  LIC, 18, 20
  particle tracing, 30
  streakball, 31
  stream arrow, 31
  streamball, 31
  streamlet, 29
  streamline, 30
  stream polygon, 31
  stream ribbon, 30
  streamrunner, 31
  stream surface, 31
  streamtube, 30
  texture advection, 24
  time surface, 31
  viewer, 97
3D IBFV, 24, 92, 119
3D and hybrid approaches, 88
  swirl motion, 89
  tumble motion, 90
3D texture advection, 24
3D viewer, 97

abstract, i
accelerated UFLIC, 20
Acknowledgements, 133
advection-diffusion, 10
affiliation, 133
analytic model, 9
animated streamline, 75, 99
  performance, 79
  summary, 116
animation, 9, 75, 78
animation time, 9
application screen shot, 106
Aristotle, 82
AUFLIC, 20
AVL, 34, 134
AVL AST, 134
axis of motion, 82

backward coordinate integration, 23
  equation, 23
Berra, Yogi, 119
blood flow on an isosurface, 69

CFD workflow manager, 97
clipping plane, 67
collaboration, 133
color mapped glyph, 89
combustion chamber, 62
comparative visualization, 15
comparison of visualization techniques, 82
comparisons, 32
computational flow simulation, 9
computation of derived data, 11
computer science, 2
conclusion, 119
contact information, 132
contouring in 2D, 27
contrast analysis, 27

contributor, 133
convergence, 11
convolution, 17
curriculum vitae, 131
curvilinear LIC, 17
CV, 131

dashed streamline, 99
dash tube, 31
data
    source of, 3
data dimensionality, 8
data source, 3, 9
data visualization, 2
da Vinci, Leonardo, 3
decoupled LIC, 22
Delft University, 134
dense, texture-based flow visualization, 6, 14
depth buffer, 101
design
    contract, 97
    object-oriented, 97
    subsystem, 97
    user interface, 101
    visualization system, 97
design of geometric and texture-based flow visualization techniques, 95
diffusion, 27
    unsteady, 27
dimensionality, 8
    data, 8
    spatial, 8
    temporal, 8
direct flow visualization, 5
    2.5D, 86
    3D, 89
    advantages, 92
    boundary surface, 86
    disadvantages, 92
    on a slice, 84, 85
distance between Graz and Vienna, 135
distance from Graz, 133
divergence, 11
DLIC, 18
dye injection, 19, 87
dye source, 101
dynamic LIC, 18

effective visualization, 119
Einstein, Albert, 34
engineering task, 72
enhanced LIC, 18
enhanced spot noise, 15
Euler, 10
Euler approximation, 5
Euler equation, 12
Euler integration, 5, 12
evaluating swirl and tumble motion, 82
evaluation of software design, 106
experimental flow visualization, 9
export button, 104

fast LIC, 17
fast LIC on surfaces, 18
fast rendering of OLIC, 19
feature-based flow visualization, 6
feature based seeding, 76
feature extraction, 7
filter kernel, 17
finite element (FE) analysis, 10
finite volume (FV) analysis, 10
FIRST, 34
    3D, 45
    advantages, 38
    algorithm, 39
    degrees of freedom (DoF), 41
    DoF, 41
    features, 38
    future work, 45
    implementation, 39
    options, 41
    performance, 43
    speed vs accuracy, 42
    summary, 110
    unsteady flow, 44
    visualization options, 42
first-order Euler, 12
first order reconstruction, 11
flexible and interactive resampling, 34
flow reconstruction, 11
flow texture, 32
flow visualization, 2, 4
    applications, 8
    classification, 5
    dense, texture-based, 6
    direct, 5
    experimental, 9
    feature based, 6
    geometric, 6
    texture-based, 6
flow visualization classification, 5
flow volume, 32
fourth order Runge-Kutta, 12
FOX toolkit, 106
FROLIC, 19
fundamentals, 10
furlike texture, 27
future work in flow visualization, 33

Gandhi, Mahandas, 47

geometric 2D flow visualization
using an integral object, 28
geometric flow visualization, 6, 27, 72
2.5D, 87
3D, 89
advantages, 92
animated streamcomet, 78
animated streamline, 75
boundary surface, 87
disadvantages, 92
dye injection, 85
future work, 81
interpretation, 74
on a slice, 85
oriented streamline, 74
pathrunner, 81
perceptual challenges, 73
results, 79
seeding plane, 75
seeding strategy, 75, 89
streakline, 85
streakrunner, 79
streamcomet, 77
streamlet, 74, 85
streamline, 74, 87
streamrunner, 77
tumble motion, 90
unsteady flow, 79
geometric flow visualization pipeline, 98
geometric flow visualization subsystem, 98
geometric flow visualization techniques, 72
summary, 114
geometric LIC, 21
geometric object
3D, 29
geometric objects on slices or boundaries, 29
glyph, 4
limitations, 4
glyph placement, 35
GPU, 14
GPU-accelerated LEA, 25
GPU-based LIC, 25
GPU-based techniques, 3, 14, 23
graphical user interface (GUI), 97
Graz, 133
Graz and Vienna, 135
grid, 10
and integration, 11
simulation, 10
types of, 10
versatile, 34
grid types, 36
GUI, 97

hedgehog visualization, 35
hierarchical LIC, 22
higher order integration, 12

IBFV, 6, 23
IBFVS, 6, 23
pipeline, 100
ideal mixing conditions, 62
image-based seeding, 76
image based flow visualization
3D, 24
image based flow visualization (IBFV), 6, 23
image based flow visualization for curved surfaces (IBFVS), 6, 23
pipeline, 100
image overlay, 101
image space advection
resampling point of view, 68
summary, 111
image space advection (IDA), 6
image space advection (ISA), 23, 47, 111
advection mesh computation, 53
backward integration, 53
boundary treatment, 53
characteristics, 47, 111
components, 50
discussion, 58, 112
dynamic case, 50, 112
edge detection, 53
future work, 58, 112
hardware accelerated vector field projection, 51
image overlay application, 56
implementation, 56
incoming boundary flow, 54
method overview, 50, 112
noise blending, 54
performance, 57
physical vs parameter vs image space, 48, 111
pipeline, 50, 100, 112
results, 57
static case, 50, 112
texture clipping, 56
time dependent mesh, 57
unsteady flow, 57
vector field de-coding, 52
vector field encoding, 51
vector field projection, 50
velocity image, 50
velocity mask, 56
vs IBFVS, 58
image space advection (ISA) and image based flow visualization
pipeline, 100
image space advection (ISA) and image based flow visualization for curved surfaces (IBFVS)

side-by-side, 100
import button, 104
in-cylinder flow, 82
information visualization, 3
Institute of Computer Graphics and Algorithms, 133
instructor, 132
intake port, 34
intake port design, 62, 82
integrated tufts, 29
integration, 11
interaction, 96
interactive seeding, 76
interactive seeding plane schematic, 76
interactive slicing, 41
interpolation, 10
investigating flow patterns with visualization, 83
investigating swirl and tumble motion, 82
conclusions, 93
future work, 93
summary, 116
trade-offs, 92
ISA, 6, 23, 47, 111
advection mesh computation, 53
backward integration, 53
boundary treatment, 53
characteristics, 47, 111
components, 50
discussion, 58, 112
dynamic case, 50, 112
edge detection, 53
future work, 58, 112
hardware accelerated vector field projection, 51
image overlay application, 56
implementation, 56
incoming boundary flow, 54
method overview, 50, 112
noise blending, 54
on an isosurface, 67
perceptual challenges, 67
performance, 57
physical vs parameter vs image space, 48, 111
pipeline, 50, 100, 112
resampling point of view, 68
results, 57
static case, 50, 112
summary, 111
texture clipping, 56
time dependent mesh, 57
unsteady flow, 57
vector field de-coding, 52
vector field encoding, 51
vector field projection, 50
velocity image, 50
velocity mask, 56
vs IBFVS, 58
ISA and IBFVS
pipeline, 100
side-by-side, 100
isosurface, 63
adaptive resolution, 63
and texture based flow visualization, 63
blood flow, 69
from CFD, 63
in 3D, 28
showing swirl motion, 63
isovalue
choosing, 91

junior researcher, 131

King, Martin Luther, 95
knowledge hierarchy, 2
Kplus, 134
Kurzfassung, ii

Lagrangian-Eulerian Advection (LEA), 25
GPU-accelerated, 25
LEA, 6, 25
lecturer, 132
LIC, 6, 14, 17
decoupled, 22
geometric, 21
GPU-based, 25
hierarchical, 22
LIC equation, 17
LIC hierarchy, 17
LIC on arbitrary surfaces, 20
LIC on surfaces, 20
LIC techniques, 14
LIC with normal component, 18
line integral convolution, 14, 17
3D, 20
accelerated, unsteady, 20
and dye injection, 19
and volume rendering, 19
curvilinear, 17
decoupled, 22
dynamic, 18
enhanced, 18
equation, 17
fast, 17
fast rendering, 19
geometric, 21
GPU-based, 25
hierarchical, 22
hierarchy, 17
multi-frequency, 19
multivariate, 18
on arbitrary surfaces, 20

on surfaces, 18, 20
oriented, 19
parallel, 18
pseudo, 22
unsteady, 17, 20
volume, 18
with normal component, 18
line integral convolution (LIC), 6
line stipple pattern, 75
Lipsa, Dan R., 2

Markov Random Field (MRF), 27
measurement
data, 9
medical visualization, 2, 3
mesh, 10
simulation, 10
types of, 10
versatile, 34
mesh manager, 97
modeling, 95
motion map, 21
moving texel, 14
moving textures, 23
MRF, 27
multi-frequency LIC, 19
multivariate LIC, 18

Navier Stokes, 10
normal mask, 65
normal mask implementation, 66
numerical integration, 5

object-oriented methodology, 97
OLIC, 19, 74
OpenGL 3D viewer, 101
optimal flow motion, 82
optimal mixing conditions, 82
oriented LIC, 19
oriented line integral convolution (OLIC), 74
oriented streamline, 74, 99
overview of thesis, 7

parallel LIC, 18
parallel spot noise, 16
particle displacement, 4
particle tracing
3D, 30
pathline, 28
pathrunner, 81
PDEs, 10
Pell, Claiborne, 72
perceptual challenges in 3D, 97
perceptual challenges with geometric flow visualization, 73
perceptual problems, 35
physical vs parameter vs image space, 48, 111
platform independence, 96
PLIC, 22
point location, 10
polar resampling, 41
pressure isosurface, 91
pseudo LIC, 22

reconstruction, 10
related dense, texture-based methods, 26
related flow visualization techniques, 14
related literature in resampling, 36
related publications, iii
related research in texture-based flow visualization, 14
related texture-based methods, 26
requirement
interaction, 96
platform independence, 96
support for simulation data, 97
support for versatile CFD grids, 97
tools that address perceptual challenges, 97
resampling
polar, 41
research assistant, 131
research vs industry, 135
Runge-Kutta, 12
higher order, 5
integrator, 5
second order, 5

saddle point, 86
scientific visualization, 2
second order gradient, 11
second order Runge-Kutta, 12
seeding plane, 75, 89
degrees of freedom (DoF), 89
seeding strategy, 75
separatrix, 87
tumble motion, 87
Seuss, Dr., 133
shaded streamline, 99
sheep, 8
simulation, 95
simulation data, 9
simulation data characteristic, 34
simulation data set, 34
simulation result manager, 98
software
design, 95
implementation, 95
software design, 95
solid streamline, 99
space

physical vs parameter vs image, 48, 111
spatial dimensionality, 8
Spot Noise, 6
spot noise, 14, 15
enhanced, 15
parallel, 16
related literature, 16
turbulent flow, 16
unsteady, 16
vs LIC, 16
spot noise equation, 15
spot noise hierarchy, 15
spot noise techniques, 14
spot noise vs LIC, 16
spray equivalence ratio, 91
steady velocity time, 9
streakball, 31
streakline, 28
streakrunner, 79
stream arrow, 31
streamball, 31
streamcomet, 77, 99
animated, 78, 99
degrees of freedom (DoF), 77
metaphor, 77
unsteady flow, 79
streamlet, 28, 74
2D, 28
3D, 29
streamline, 28, 74
2D, 28
3D, 30
animated, 75, 99
dashed, 99
efficient construction, 30
oriented, 74, 99
shaded, 99
solid, 99
stipple pattern, 75
via dye injection, 87
streamline, oriented streamline, streamlet, streamcomet
side-by-side, 114
streamline renderer, 99
streamline seeding, 28
2D, 28
boundary surfaces, 29
multiresolution, 29
stream polygon, 31
stream ribbon, 30
efficient construction, 30
streamrunner, 31, 77
stream surface, 31
streamtube, 30
efficient construction, 30
subsystem
design and implementation, 98
geometric flow visualization, 98
texture-based flow visualization, 98, 100
summary
animated streamline, 116
FIRST, 110
geometric flow visualization techniques, 114
image space advection, 111
investigating swirl and tumble motion, 116
ISA, 111
oriented streamline, 115
resampler, 110
streamcomet, 115
streamlet, 115
texture-based flow visualization on isosurfaces,
113
thesis, 109
summary of thesis, 109
support for simulation data, 97
support for versatile CFD grids, 97
surface particles, 31
swirl and tumble motion, 82
swirl flow, 62
swirl motion, 62, 82
2.5D, 86, 88
3D, 89
and an isosurface, 91
boundary surface, 86, 88
direct flow visualization, 86
geometric flow visualization, 89
on a slice, 84, 85
streamlet, 85
swirl pattern, 62
Swiss Center for Scientific Computing, 133
system requirements and goals, 96

task of engineer, 72
Technische Universiteit Eindhoven, 134
temporal dimensionality, 8
Texas, 8
texture-based flow visualization, 6
3D, 92
advantages, 92
disadvantages, 92
evolution, 119
texture-based flow visualization on isosurfaces
summary, 113
texture-based flow visualization subsystem, 98, 100
texture-based flow visualizer, 101
texture advection, 14, 23
3D, 24
texture based flow visualization
2.5D, 88

3D a hybrid approach, 91
and an isosurface, 63, 91
boundary surface, 88
on a slice, 86
resampling point of view, 68
swirl motion, 88
tumble motion, 88
texture based flow visualization on a boundary surface, 65
texture based flow visualization on an isosurface, 64
advantages, 63
applying a normal mask, 65
a resampling point of view, 68
blood flow, 69
clipping plane, 67
future work, 70
method background, 64
occlusion, 67
perceptual challenges, 67
performance, 70
results, 69
slices, 67
visual comparison, 67
texture splat, 26
texture stack, 101
texture transport, 26
thesis
conclusion, 119
thesis summary, 109
three dimensional LIC, 20
timeline, 28, 87
tumble motion, 87
time surface, 31
3D, 31
Tolkien, J. R. R., 109
topological based seeding, 76
trade-off in visualization, 92
triangle packing, 49, 112
tumble and swirl motion, 82
tumble motion, 82
2.5D, 86, 88
3D and hybrid approach, 90
and isosurface, 91
boundary surface, 86
direction flow visualization, 86
dye injection, 85, 86
geometric flow visualization, 87, 90
non-ideal, 88
on a slice, 85, 86
saddle point, 86
separatrix, 87
streakline, 85
streamline, 87
timeline, 87
UFAC, 26
UFLIC, 119
UI design, 101
ULIC, 20
uncertainty visualization, 33
University of New Hampshire, 131, 133
University of Pau, 133
University of Stuttgart, 134
unsteady diffusion, 27
unsteady flow
LIC, 20
unsteady flow advection convolution (UFAC), 26
unsteady LIC, 17, 20
unsteady spot noise, 16
unsteady velocity time, 9
user community, 102
user controlled glyph placement, 35
user controlled resolution, 35
user interface
description, 104
design, 104
screen shot, 103
user interface design, 101
user study, 32

vector field visualization, 9
velocity, 4
velocity data, 4
velocity image, 101
velocity mask, 65
versatile grid, 34
versatile mesh, 34
versatility of CFD meshes, 72
Vienna and Graz, 135
Vinci, Leonardo da, 3
visualization, 2, 95
data, 2
definition, 2
flow, 2
information, 3
medical, 2, 3
scientific, 2
trade-off, 92
visualization of uncertainty, 33
visualization system design, 97
visualizing flow motion
2D slices, 84
volume LIC, 18
volume of email, 133
volume rendering, 3
VRVis Research Center, 131, 133