Feature Extraction and Visualisation of Flow Fields

Frits H. Post, Benjamin Vrolijk
{F.H.Post,B.Vrolijk}@its.tudelft.nl
Delft University of Technology, The Netherlands
http://visualisation.tudelft.nl/

and

Helwig Hauser, Robert S. Laramee, Helmut Doleisch
{Hauser,Laramee,Doleisch}@vrvis.at
VRVis Research Center, Austria
http://www.vrvis.at/

Abstract

Flow visualisation has already been a very attractive part of visualisation research for a long time. Usually very large data sets need to be processed, which often consist of multivariate data with a large number of sample locations, often arranged in multiple time steps. Recently, the steadily increasing performance of computers again has become a driving factor for a new boom in flow visualisation, especially in techniques based on feature extraction, vector field clustering, and topology extraction.
In this state-of-the-art report, an attempt was made to (1) provide a useful categorisation of FlowVis solutions, (2) give an overview of existing solutions, and (3) focus on recent work, especially in the field of feature extraction. In separate sections we describe (a) direct visualisation techniques such as hedgehog plots, (b) visualisation using integral objects, such as streamlines, (c) texture-based techniques, including spot noise and line integral convolution, and (d) techniques based on extraction of features or flow topology.

Categories and Subject Descriptors (according to ACM CCS): I.3 [Computer Graphics]: visualisation, flow visualisation, computational flow visualisation

1. Introduction

Computers have become increasingly important in many aspects of society — in science, business and economics, education and politics, as well as in many other fields, computers are used to acquire, store, process, and communicate data, not in the least to users. *Visualisation*, as a separate field of research and development in computer science, addresses exactly this bridge between data and user: visualisation solutions help users to explore, analyse, and present their data.

In *flow visualisation* (FlowVis) — one of the traditional subfields of visualisation — a rich variety of application fields is given, form the automotive industry, aerodynamics, turbomachinery design, weather simulation and meteorology, climate modelling, ground water flow, medical applications, etc., with significantly different characteristics relating to the data and user goals. Consequently, the spectrum of FlowVis solutions is very rich, spanning multiple dimensions of technical aspects, e.g., 2D vs. 3D solutions, techniques for steady and time-dependent data, et cetera.

1.1. Aspects of Flow Visualisation

Bringing many of those solutions in a linear order (as necessary for a text like this), is not at all easy or intuitive. Several options of subdividing this broad field of literature are possible. Hesselink et al., for example, addressed the difficult problem of how to categorise FlowVis techniques in their 1994 overview of (at that time) recent research issues [33]. In the following subsections several of those aspects are discussed on a higher level, before literature is addressed directly later.

Direct vs. integration-based vs. feature-based flow visualisation

According to the different needs of the users there are different approaches to flow visualisation (cf. Figure 1c). One is to do *direct flow visualisation* by using an as direct as possible translation of the flow data into visualisation cues, such as by drawing arrows. FlowVis solutions of this kind allow immediate investigation of the flow data, without a lot of mental translation effort.

For a better communication of the long-term behaviour induced by flow dynamics, *integration-based approaches* first

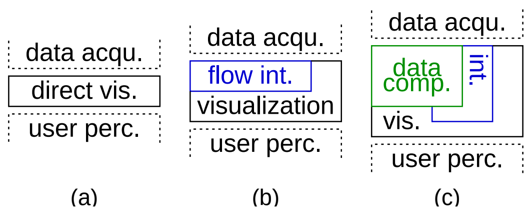


Figure 1: *Direct flow visualisation (a) vs. FlowVis based on flow integration (b) vs. FlowVis based on derived data such as flow features or flow topology (c). This classification relates to the first-level structure of this report.*

integrate the flow data and use resulting integral objects as basis for visualisation, e.g., using streamlines for visualisation.

Another approach for visualising flow data is the *feature-based approach*, in which an abstraction step is performed first. From the original data set, interesting objects are extracted, such as important phenomena or topological information of the flow. These flow features are an abstraction of the data, and can be visualised efficiently and without the original data. Because the original data is not needed anymore, a huge data reduction is achieved, of a factor 1000 or more. This makes this approach very suitable for large (time-dependent) data sets, originating from computational fluid dynamics simulations. These data sets are simply too large to visualise directly, and therefore, a lot of time is required in preprocessing, for computing the features (feature extraction). But once this preprocessing has been performed, visualisation can be done very quickly.

In this overview we use separate chapters for the aforementioned classes of approaches: direct flow visualisation is discussed in Section 2, integration-based FlowVis in Sections 3 and 4, and feature-based FlowVis is described in Sections 5 through 8. Figure 1 illustrates the difference between the aforementioned classes — note the increasing amount of computation spent within the visualisation step when changing from direct FlowVis (a) to feature-based FlowVis (c).

Spatial dimensions vs. time

In flow visualisation, available solutions significantly differ with respect to the given dimensionality of the flow data. Techniques which are useful for 2D data, like colour coding or arrow plots, sometimes lack similar advantages in 3D. Also, the question, whether the flow data is steady or time-dependent, usually makes a big difference with respect to the FlowVis solution of choice.

In this state-of-the-art report, we (at least partially) substructure the sections about different classes of FlowVis solutions into subsections with respect to different spatial dimensions involved. Although there are lots of interesting works about 1D FlowVis as well as nD FlowVis (with $n > 3$), this report clearly focuses on two and three spatial dimensions.

Below, the top-level sections start with a subsection on *2D FlowVis techniques* (Sections n.1), i.e., covering solutions which focus on 2D flow data (in 2D domains). Since the 2D domain inherently corresponds to the 2D screen, good overviews are possible for these kinds of techniques like with the use of 2D LIC (see below for details). However, the reader should be aware, that real-world flows (at least when talking about fluids or gases) are rarely two-dimensional — data sets therefore are often slices out of a stack of those, or stem from simplifications of the underlying model.

A second subsection (Sections n.2) discusses *FlowVis solutions for boundary flows or sectional subsets of 3D flows*, for example, flow data on planar cross sections. This subsection therefore deals with 2D flow data, at least with respect to the local dimensionality of the data, but which is embedded within 3D space. Whereas boundary flows often are primarily interesting to the user anyway (for example in aerospace design), the visualisation of sectional subsets of 3D flow usually needs special care (not at the least because of the usually missing third flow component). Especially the use of integral curves across flow cross sections is questionable as the suggested particle paths (in general) do not correspond to actual flow trajectories which naturally extend to 3D in this case.

Finally, a third subsection (Sections n.3) discusses *truly 3D FlowVis solutions*, i.e., visualisation techniques, which apply to true 3D flow data. With true 3D FlowVis, rendering becomes a central issue — in many cases compromises are needed, trading visibility for completeness. Solutions range from clipping and opacity modulations to feature-based selections.

In addition to the spatial dimensions as addressed above, also *dimensionality with respect to time* is of great importance in flow visualisation. First of all, flow data itself incorporates a notion of time — flows often are interpreted as differential data with respect to time, i.e., when integrating the data, paths of moving entities are obtained. Additionally, the flow itself can change over time (like in turbulent flows, for example), resulting in time-dependent or unsteady data. In this case, two dimensions of time are present and the visualisation must carefully distinguish between both in order to prevent the user from being confused. This is especially true, when animation should be used for flow visualisation. Then, even a third temporal dimension can show up in a visualisation, requiring special care to avoid confusion along with interpretation of the animations.

Although the distinction between steady and unsteady flows could open another dimension when sorting FlowVis literature, in this report solutions for time-dependent data are put beside related techniques for steady data.

Computational vs. experimental and empirical FlowVis

Flow visualisation, as discussed in this literature overview, is considered to be equivalent to what others call *computational flow visualisation* — just to distinguish it from the large and old fields of experimental and empirical flow visualisation.

Although we do not have space to also focus on those other variants of FlowVis, it is interesting to recognise that many computational FlowVis solutions more or less mimic the visual appearance of well-accepted techniques in experimental visualisation (cf. particle traces, dye injection, et cetera).

Data from simulation vs. measurements or models

Computational FlowVis, in general, deals with data that exhibit temporal dynamics such as results from flow simulation (e.g., the simulation of fluid flow through a turbine), flow measurements (possibly acquired through laser-based technology), or analytic models of flows (e.g., dynamical systems, given as set of differential equations).

In this report we mainly focus on flow visualisation dealing with data from flow simulation, i.e., flow data given as a set of samples on some kind of grid, whereas solutions for data from flow measurements or flow modelling are only addressed in less detail. Technical issues frequently arise due to the combination of extremely large data sets and demanding user requirements such as interactive visualisation of time-dependent data. Therefore, solutions in the field of parallel computing [11, 60, 138, 170], out-of-core rendering [147], and rendering of compressed data [166] are often discussed in the FlowVis literature.

Placement and interaction

Many FlowVis solutions build on the use of individual visualisation objects, for example, streamlines. For at least three reasons, the placement of those visualisation cues is an issue within FlowVis literature: (1) when using integral objects such as streamlines, an even distribution of seed locations usually does not result in an even distribution of integral objects — separate algorithms need to be employed; (2) when dealing with 3D flow data, occlusion and/or visualisation complexity raises special challenges — dense placement often results in severe cluttering within rendered images; (3) when using feature-based strategies, placement needs to be coupled (and aligned) with the feature extraction parts of the visualisation.

In addition to placement, user interaction plays an important role, especially in case of flow analysis. Users require systems which allow (1) navigation, including zooming, panning, etc., (2) interactive placement of visualisation cues, for example, using an interactive rake for streamline seeding, as well as other means to influence the visualisation, or even (3) options of interacting with the flow data, for example, through steering.

Last but not least *human-computer interaction* challenges present themselves throughout flow visualisation research, especially in the categories of perception in 3D, and interaction. For there is strong evidence that both 3D visualisation [154] and interaction [34] are very important components for the user in understanding the data.

1.2. FlowVis Fundamentals

Before outlining some of the most important FlowVis techniques in the main part of this paper, a short overview about the common mathematical background as well as some general concepts with regard to the computation of FlowVis results are discussed.

Flow data

An inherent characteristic of flow data is that derivative information is given with respect to time, which is laid out across an n-dimensional domain $\Omega \subset R^n$, for example, for representing 3D fluid flow ($n = 3$). In the case of multidimensional flow data ($n > 1$), temporal derivatives $\mathbf{v}$ of nD locations $\mathbf{p}$ within the flow domain Ω are n-dimensional vectors:

$$\mathbf{v} = \mathrm{d}\mathbf{p}/\mathrm{d}t, \quad \mathbf{p} \in \Omega \subseteq R^n,\ \mathbf{v} \in R^n,\ t \in R \qquad (1)$$

In analytic models (like dynamical systems), vectors $\mathbf{v}$ often are described as functions of the respective spatial locations $\mathbf{p}$, say like $\mathbf{v} = \mathbf{Ap}$ for steady linear flow data if $\mathbf{A}$ is a constant $n{\times}n$-matrix. A general formulation of (possibly unsteady, i.e., time-dependent) flow data $\mathbf{v}$ would be

$$\mathbf{v}(\mathbf{p}, t) : \Omega \times \Pi \to R^n \qquad (2)$$

where $\mathbf{p} \in \Omega \subset R^n$ represents the spatial reference of the flow and $t \in \Pi \subset R$ represents the system time. If t is considered to be constant, i.e., for steady flow data, the more simple case of $\mathbf{v}(\mathbf{p}) : \Omega \to R^n$ is given.

In cases of results from nD flow simulation, like in automotive applications or airplane design, vector data $\mathbf{v}$ usually is not given in analytic form, but needs to be reconstructed from the (discrete) simulation output. As usually numerical methods are used to actually do the flow simulation such as finite element methods. The output of flow simulation usually is a large-sized grid of many sample vectors $\mathbf{v}_{i,t}$, which discretely represent the solution of the simulation process (at time steps t). For further procedure, it is assumed that the flow simulation was based on an (at least locally) continuous model of the flow, thus allowing for continuous reconstruction of the flow data $\mathbf{v}$ during visualisation. One option for doing so would be to apply a reconstruction filter $h : R^n \to R$ to compute $\mathbf{v}(\mathbf{p}, t) = \sum_i h(\mathbf{p} - \mathbf{p}_i)\mathbf{v}_{i,t}$. As — for practical reasons — filter h usually has only local extent (around the origin), efficient procedures for finding those flow samples $\mathbf{v}_{i,t}$, which are nearest to the query point $\mathbf{p}$, are needed to do proper reconstruction.

Grids

In flow simulation, the vector samples $\mathbf{v}_{i,t}$ usually are laid out across the flow domain according to a certain type of grid. Grid types range from simple Cartesian grids over curvilinear grids to complex unstructured grids. Typically, simulation grids also exhibit large variations in cell sizes. This variety of grids stems from the high influence of grid design onto the flow simulation process and the thereby derived need to model the flow grid as optimal as possible with respect to the simulation in mind.

Although the principal theory of function reconstruction from discrete samples does not exhibit too many differences with respect to grid types involved, the practical handling does. While neighbour searching might be trivial in a Cartesian grid, it usually is not in a tetrahedral grid. Similar differences are given for the problems of point location and vector reconstruction. In the following we shortly describe several fundamental operations which form the basis for FlowVis computations on simulation grids.

Starting with *point location*, i.e., the problem of finding the grid cell which a given nD-point lies in, usually two cases are distinguished. For the general point location problem special data structures can be used which subdivide the spatial domain to speed up the search. The second case of iterative point location, which often is needed during integral curve computation, usually allows for quite efficient algorithms due to exploitation of spatial coherence. One kind of algorithm starts with an initial guess for the target cell, checks for containment then and refining accordingly afterwards. This process is iterated until the target cell is found. More details can be found in older texts about flow visualisation fundamentals [129, 95].

Beside point location, *flow reconstruction* within a cell of the flow data set is a crucial issue in flow visualisation. Often, once the cell which contains the query location is found, only the sample vectors at the cell's vertices are considered for flow reconstruction. The most often used approach is first-order reconstruction by performing linear interpolations within the cell. Within a hexahedral cell in 3D, for example, trilinear flow reconstruction can be used.

Using point location and flow reconstruction, flow visualisation can already start: vectors can be represented (for example, by arrows), virtual particles can be injected and traced across the flow domain. Nevertheless, the *computation of derived data* might be necessary to do more sophisticated FlowVis. Usually, 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, flow rotation and shear, et cetera. For feature extraction, also 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, 77].

Flow 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 (long-term) evolution induced by the flow data. An example would be flow visualisation by the use of particle advection. A respective particle path $\mathbf{p}(s)$ — here through unsteady flow — would be defined by

$$\mathbf{p}(s) = \mathbf{p}_0 + \int_{\tau=0}^{s} \mathbf{v}(\mathbf{p}(\tau), \tau + t_0)\, \mathrm{d}\tau \qquad (3)$$

where $\mathbf{p}_0$ represents the seed location of the particle path and t_0 equals the time when the particle was seeded. Note, that Equations 2 and 3 are more or less complimentary to each other. For other types of integral curves such as streamlines, streaklines, etc., refer to later parts of this text or previous works [129, 61].

In addition to the theoretical specification of integral curves, it is important to note, that respective integral equations like Equation 3 usually cannot be resolved for the curve function analytically, and thereby *numerical integration methods* need to be employed. The most simple approach is to use a first-order Euler method to compute an approximation $\mathbf{p_E}$ — one iteration of the curve integration is specified as by

$$\mathbf{p}_E(t + \Delta t) = \mathbf{p}(t) + \Delta t\, \mathbf{v}(\mathbf{p}(t), t) \qquad (4)$$

where Δt usually is 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, which uses the Euler approximation $\mathbf{p}_E$ as a look-ahead to compute a better approximation $\mathbf{p}_{RK2}$ of the integral curve:

$$\begin{aligned}\mathbf{p}_{RK2}(t + \Delta t) &= \\ &\mathbf{p}(t) + \Delta t \cdot (\mathbf{v}(\mathbf{p}(t), t) + \mathbf{v}(\mathbf{p}_E(t + \Delta t), t))/2 \qquad (5)\end{aligned}$$

Higher-order methods like the often used fourth-order Runge-Kutta integrator utilise more such steps to better approximate the local behaviour of the integral curve. Also, adaptive step sizes are used to make smaller steps in regions where lots of changes take place in the flow.

In the following, four classes of approaches in the field of flow visualisation are discussed — direct flow visualisation is described in Section 2, texture-based FlowVis in Section 3, geometric FlowVis is discussed in Section 4 and finally, feature-based flow visualisation is described in Sections 5 through 8.

2. Direct flow visualisation

Direct, or global, flow visualisation 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 direct, without complex conversion or

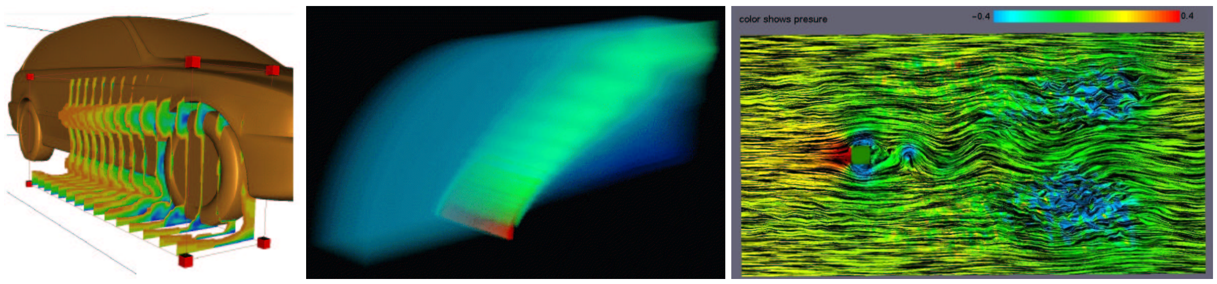


Figure 2: *Examples of direct flow visualisation — an interactive slicing probe with coloured slices and scalar clipping (left)* [122]*; direct volume rendering based on resampling (middle)* [160]*; texture-based, coloured spot noise (right)* [65].

extraction steps. These techniques are perhaps the most intuitive visualisation 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 visualisation results.

2.1. Direct FlowVis in 2D

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

Colour coding in 2D

A common direct flow visualisation technique is to map flow attributes such as velocity, pressure, or temperature to colour. Since colour plots are widely distributed, this approach results in very intuitive depictions. Of course, the colour scale which is used for mapping must be chosen carefully with respect to perceptual differentiation.

Colour coding for 2D FlowVis extends to time-dependent data very well, resulting in moving colour plots according to changes of the flow properties over time.

Arrow plots in 2D

A natural vector visualisation technique is to map a line, arrow, or glyph to each sample point in the field, oriented according to the flow field, as in Figure 6 (left). 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) normalised arrows of unit length which visualise the direction of the flow only and (2) arrows of varying length that is proportional to the flow velocity. Klassen and Harrington [59] and Schroeder et al. [121] call this technique a *hedgehog visualisation* (because of the bristly result).

2D hedgehog plots can be extended to time-dependent data, although bigger time steps might result in jumping arrows, diminishing the quality of such a visualisation.

Hybrid direct FlowVis in 2D

Kirby et al. propose simultaneous visualisation of multiple values (of 2D flow data) by using a layering concept related to the painting process of artists [57]. Arrow plots are mixed with colour coding to provide visualisation results rich of information.

2.2. Direct FlowVis on slices or boundaries

When dealing with 3D flow data, visualisation naturally faces additional challenges such as 3D rendering. Acting as a middle ground between 2D FlowVis and the visualisation 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 FlowVis usually 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.

Colour coding on slices or boundaries

Colour coding is very effective for visualising boundary flows or sectional subsets of 3D flow data. A good example is NASA's Field Encapsulation Library [85], which allows to easily use both techniques for various flow data.

Schulz et al. also use colour coding of scalars on 2D slices in 3D automotive simulation data [122] as shown in Figure 2 (left). 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 (coloured) 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 visualisation technique [19]. However, results of such a visualisation should be interpreted carefully, as flow components which are orthogonal to the slice are usually not depicted.

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 visualisation [145].

2.3. Direct FlowVis in 3D

After discussing direct FlowVis on slices and boundary surfaces, direct FlowVis of real 3D flows is discussed in this subsection. 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 FlowVis

The natural extension of colour coding in 2D (or on slices, etc.) is colour 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 visualisation, i.e., volume visualisation. However, those challenges, which closely correspond to flow visualisation 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 tricky on those kinds of grids, 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. [15], as well as Ebert et al. [18] applied volume rendering techniques to vector fields. Little later, Frühauf applied ray casting to vector fields [22]. Recently, Westermann, presented a relatively fast 3D volume rendering method using a resampling technique for vector field data from unstructured to Cartesian grids [160]. A result from this technique is illustrated in Figure 2 (middle).

Recently, Clyne and Dennis [14] as well as Glau [24] 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 visualise thermal flows in the passenger compartment of an automobile [90]. 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 visualisation in a system that supports computational steering [137]. Their visualisation results are extended to the CAVE environment.

Recently, Ebert and Rheingans demonstrated the use of nonphotorealistic volume rendering techniques for 3D flow data [17]. They apply, for example, 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 [8]. 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.

3. Texture-based Visualisation

We make a distinction between geometric flow visualisation (see Section 4) and dense, texture-based flow visualisation, however, these two topics are closely coupled: conceptually, the path from using geometric objects to texture-based visualisation 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. Likewise, the path from dense, texture-based visualisation to visualisation using geometric objects is obtained using something such as a sparse texture for texture advection.

Texture-based techniques in flow visualisation can provide dense spatial resolution images. Texture-based algorithms are effective, versatile, and applicable to a wide spectrum of applications. Sanna et al. present a summary of this research in a survey paper [117].

3.1. Texture-based FlowVis in 2D

In this subsection, we describe texture-based FlowVis solutions for 2D flow data, i.e., spot noise, line integral convolution (LIC), and related approaches.

Spot noise in 2D

Spot noise, introduced by Van Wijk [162], was amongst the first texture-based techniques for vector field visualisation. Spot noise generates a texture by distributing a set of intensity functions, or spots, over the domain. Each spot represents a particle moving over a small step in time and results in a streak in the direction of the local flow from where the particle is seeded.

One limitation of the original spot noise algorithm was the lack of velocity magnitude information in the resulting texture. Enhanced spot noise [70], by De Leeuw and Van Wijk was introduced to address this problem. Spot noise has also been applied to the visualisation of turbulent flow [67] by De Leeuw et al. A spot noise algorithm for interactive visualisation is proposed by De Leeuw [65], also. De Leeuw and Van Liere also compare spot noise to LIC [68]. Spot noise in 2D combined with colour coding is shown in Figure 2 (right).

Line integral convolution in 2D

Line integral convolution (LIC), first introduced by Cabral and Leedom [12] is a very popular technique for the dense coverage of vector fields with flow visualisation cues. The original methodology behind LIC takes as input a vector field on a Cartesian grid and a white noise texture of the same size. The noise texture is locally filtered (smoothed) along the path of streamlines to acquire a dense visualisation of the flow field. See Figure 6 (middle) for an example.

The research in flow visualisation based on LIC described here extends LIC in several ways: (1) adding directional cues, (2) showing velocity magnitudes, (3) added support for non-Cartesian grids, (4) allowing real-time and interactive exploration, (5) extending LIC to 3D, and (6) extending LIC to unsteady vector field visualisation with time coherency.

Shen et al. address the problem of directional cues in LIC by combining animation and introducing dye advection into the computation [126]. Kiu and Banks proposed to use a multifrequency noise for LIC [58]. The spatial frequency of the noise is a function of the magnitude of the local velocity in the field.

Khouas et al. synthesise LIC-like images in 2D with furlike textures [56]. Their technique is able to locally control attributes of the output texture such as orientation, length, density, and colour.

Much research has been dedicated to bringing LIC computation to interactive rates. Stalling and Hege present significant improvements in LIC performance by exploiting coherence along streamlines [135, 29]. Parallel implementations of LIC are presented by Cabral and Leedom [11], and Zöckler et al. [170].

OLIC for 2D FlowVis

Wegenkittl et al. also address the problem of orientation of flow with their OLIC (Oriented Line Integral Convolution) approach [157]. Conceptually, the OLIC algorithm makes use of a sparse texture consisting of many separated spots which are more or less smeared in the direction of the local vector field through integration. A fast version of OLIC (called FROLIC) is presented by Wegenkittl and Gröller [156] via a trade-off of accuracy for time. Berger and Gröller present an algorithm for animating 2D FROLIC images over the world wide web [7].

Löffelmann et al. use virtual ink droplets, like streamlets, to visualise 2D dynamical systems [74]. Similar to oriented line integral convolution (OLIC), the virtual ink droplet method is capable of visualising not only direction and velocity of flow, but also the orientation of vectors. See Figure 6 for a comparison between streamlets (right) and LIC (middle).

2D Texture Advection

Jobard and Lefer use a motion map data structure for animating 2D, steady-state flow fields [47]. The motion map contains both a dense representation of the flow and the information required 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.

Jobard et al. propose a technique to visualise dense representations of unsteady vector fields based on what they call a Lagrangian-Eulerian Advection scheme [45]. The algorithm combines a dense, time-dependent, integration-based representation of the vector field with interactive frame rates. Some results from the technique are shown in Figure 3.

Unsteady flow visualisation techniques may address the problem of interactive performance time through the use of texture mapping supported by the graphics hardware. Becker and Rumpf illustrate hardware-supported texture transport for 2D, unsteady flow data [6].

Jobard et al. [43, 44] present additional 2D, unsteady flow visualisation techniques. They achieve high performance via the use of graphics hardware. They also detail spatial and temporal coherence techniques, dye advection techniques, and feature extraction.

3.2. Texture-based FlowVis on surfaces or boundaries

Texture-based techniques are, in general, better methods for conveying flow information on sectional slices than techniques using (long) geometric objects. This is because the connection along the path of what would be a streamline is lost with dense, texture-based techniques. Thus the depiction of the flow is not misleading in terms of a potential suggestions of particle paths. Let us recall that the vector component orthogonal to the slice is removed when using texture-based and geometric methods for visualisation results.

Spot noise on boundaries or slices

De Leeuw et al. extend the spot noise algorithm to surfaces in a study that compares experimental surface flow visualisation (with oil) to that of spot noise on surfaces [66].

A combination of both texture-based FlowVis (on slices) and 3D arrows for 3D FlowVis is employed by Telea and Van Wijk [144] where arrows denote the main characteristics of the 3D flow (after clustering) and a 2D slice with spot noise or

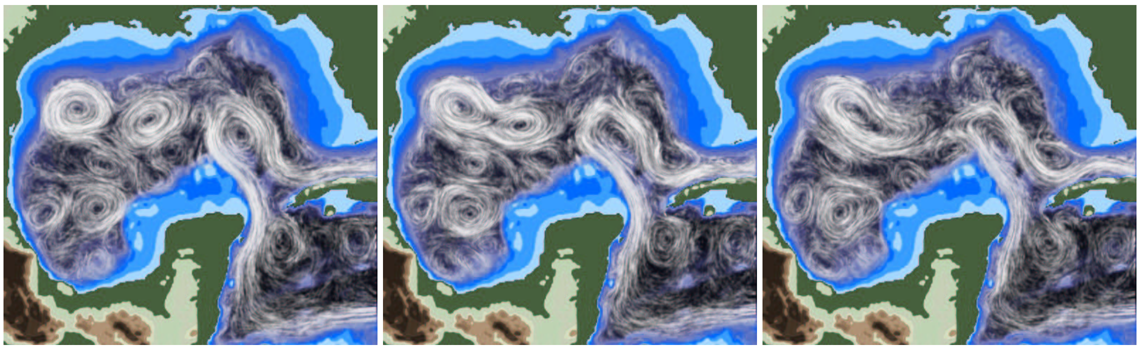


Figure 3: *Three images taken from an animation of an unsteady vector field created with the Lagrangian-Eulerian advection algorithm* [45].

LIC is used to visualise the rest of the vector field (on a slice only).

LIC for boundary flows

A large body of research literature is dedicated to the extension of LIC onto boundary surfaces, surveyed, for example, by Stalling [134].

The extension of LIC to non-Cartesian grids and surfaces is presented by researchers such as Forssell [20]. Forssell and Cohen [21] extend LIC to curvilinear surfaces with animation techniques, add magnitude and direction information, and show how to use LIC to depict time-dependent flows. Their algorithm also utilises texture mapping hardware to improve performance time towards interactive rates.

Teitzel et al. [142] present an approach that works on both 2D unstructured grids and directly on triangulated surfaces in three-dimensional space. Mao et al. [82] present an algorithm for convolving solid white noise on triangle meshes in 3D space, and extend LIC for visualising a vector field on arbitrary 3D surfaces.

Battke et al. [4] describe an extension of LIC for arbitrary surfaces in 3D. Some approaches are limited to curvilinear surfaces, i.e., surfaces which can be parameterised by using 2D-coordinates. Their method also handles the case of general, multiply connected surfaces.

Scheuermann et al. present a method for visualising 3D vector fields that are defined on a 3D manifold [118]. Their work addresses the normal vector component to the surface that other methods do not.

A 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 multigranularity noise as the input image for LIC [81].

UFLIC, PLIC, et cetera

Shen and Kao present UFLIC (Unsteady Flow LIC), which incorporates time into the convolution [127, 125]. See Figure 4 (left). Their algorithm addresses problems with temporal coherency by successively updating the convolution results over time. They also propose a parallel UFLIC algorithm.

Verma et al. present a method for comparative analysis of streamlines and LIC called PLIC [149]. A visual comparison between ELIC (enhanced LIC) [89], PLIC, and UFLIC is shown in Figure 4.

3.3. Texture-based FlowVis in 3D

High computational costs, demanding memory requirements, occlusion, and visual complexity can all be inhibitants for texture-based flow visualisation in 3D.

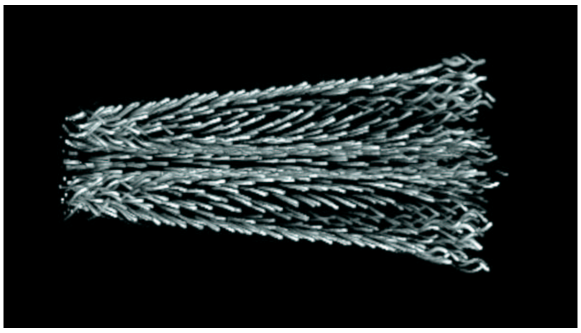


Figure 5: *3D LIC* [38].

LIC in 3D

Occlusion and interactive performance are nontrivial challenges when implementing LIC in 3D (shown in Figure 5). Rezk-Salama et al. tackle the problem of interactive performance using a 3D-texture mapping approach combined with

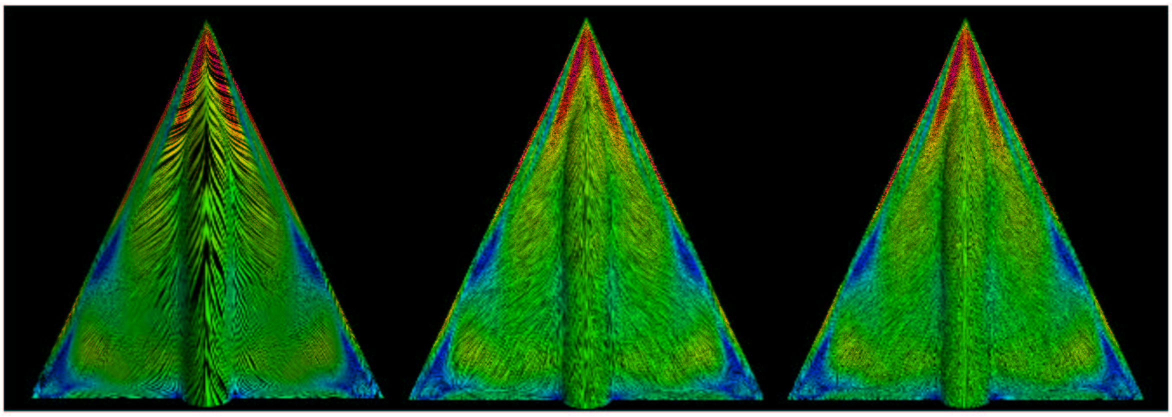


Figure 4: *A comparison of 3 LIC techniques: (left) UFLIC, (middle) ELIC, and (right) PLIC* [149].

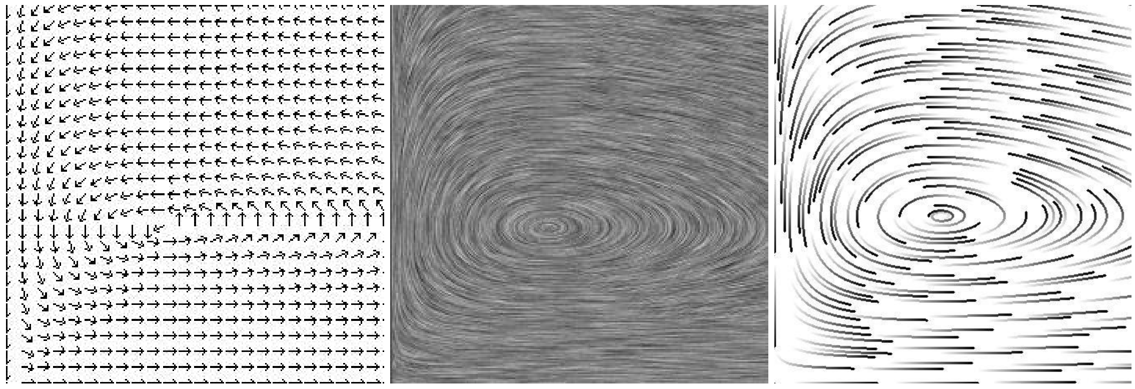


Figure 6: *Example of comparing FlowVis techniques from Sections 2, 3, and 4* [72]*. FlowVis by the use of arrows (left) is compared to texture-based FlowVis by the use of LIC (middle) and FlowVis based on geometric objects (right).*

an interactive clipping plane to address the problems of occlusion and interaction [103].

A combined approach of direct volume rendering and LIC is taken by Interrante [40] for extending LIC to 3D. Interrante and Grosch address some perceptual difficulties encountered with dense, 3D visualisations [38, 39, 40]. Techniques for selectively emphasising important regions of interest in the flow, enhancing depth perception, and improving orientation perception of overlapping streamlines are discussed.

Texture advection in 3D

Kao et al. discuss the use of 3D and 4D texture advection for the visualisation of 3D fluid flows [51]. Formidable challenges are introduced by the memory requirements involved in using 3D and 4D textures. They also apply a steady-state animation to these 3D and 4D textures.

4. Geometric Flow Visualisation

Geometric FlowVis entails extracting geometric objects for which their shape is directly related to the underlying data. In what follows, we discuss geometric flow visualisation 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 colour 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 colour plot, transitions are shown by a change of colour. With contouring, an explicit line or curve is drawn.

Isosurfaces for 3D FlowVis

Extending contouring from 2D to 3D, results in the use of isosurfaces for 3D flow visualisation. Special care needs to be taken with isovalue selection, mostly because of the usually 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 visualisation of shock waves [139] or burning fronts in simulated combustion data. Furthermore, when scalar clipping is used together with colour 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 visualisation [109]. Treinish applies isosurfacing to visualise (unsteady) weather data [145]. Weber et al. [155] present crack-free isosurface extraction for adaptive (multiresolution) grids. Laramee and Bergeron provide isosurfaces for super adaptive resolution grids [63].

4.1. 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. Also, the seeding problem is addressed, which requires a solution in order to realise better distributions of integral objects.

Streamlets 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. Figure 6 shows an example, where several streamlets are used to visualise a 2D flow field.

Streamlines in 2D

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 visualisation. A *pathline* or *particle trace* is the trajectory that a particle follows in a fluid flow [121]. 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 [88]. A *streakline* is traced by a set of particles that have previously passed through a unique point in the domain [121]. Streaklines relate to continuous injection of foreign material into real flow. Sanna et al. present an adaptive visualisation method using streaklines where the seeding of streaklines is a function of local vorticity [116].

Streamline seeding in 2D

One important aspect of streamlines, or integral curves, when used for visualising 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 [146] as well as Jobard and Lefer [46] 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. [150] 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 visualising large, 2D, steady-state vector fields [49]. 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 visualisations of the vector field at the streamline level [48]. 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.

4.2. 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, again.

Integrated tufts

Wegenkittl et al. use *integrated tufts* (similar to streamlets), seeded on specific equilibrium surfaces, for the visualisation of a complex dynamical system [158], 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 visualising boundary flows or sectional slices through 3D flow [19]. 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.

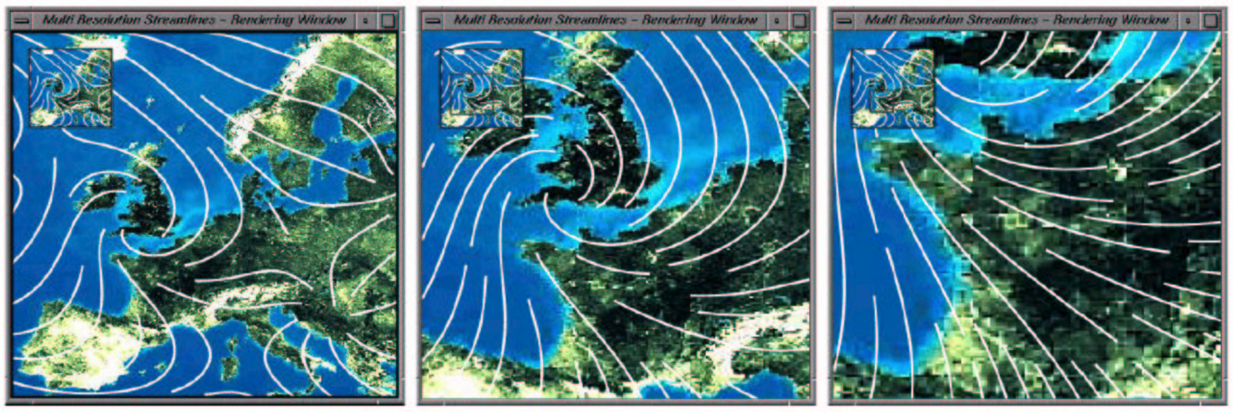

Multi Resolution Streamlines – Rendering Window
Multi Resolution Streamlines – Rendering Window
Multi Resolution Streamlines – Rendering Window

Figure 7: *Three images from an interactive exploration of a vector field using the MR viewer* [49]. *A suitable level of resolution can be chosen while maintaining a roughly constant streamline density.*

Streamline seeding on boundary surfaces

Mao et al. [80] extend the streamline seeding of Turk and Banks [146] in order to generate evenly distributed streamlines on boundary surfaces within curvilinear grids.

4.3. 3D FlowVis using geometric objects

When dealing with 3D flow, a rich variety of geometric objects is available for flow visualisation. 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 visualisation cues [73].

Streamlines in 3D

At NASA the Flow Analysis Software Toolkit (FAST) [1] is used to visualise 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 [169]. Their shading technique increases depth information. By making the streamlines partially transparent, they also address the problem of occlusion, as shown in Figure 8 (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 [55]. 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 [141, 143]. 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 visualise huge data sets.

Nielson presents efficient and accurate methods for computing tangent curves for 3D flows [87]. 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 Sadarjoen et al. [111].

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 [10] demonstrate seeding of integral objects in a virtual 3D environment by use of a so-called *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 visualise rotational behaviour of the 3D flow (which is not possible with streamlines alone) [148]. A streamtube is a thick streamline that can be extended to show the expansion of the flow [148]. Stream ribbons and streamtubes offer advantages over streamlines in that way, 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 [148]. A specialised 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 [23] use so-called *dash tubes*, i.e., animated, opacity-mapped streamtubes, as a visualisation 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 *stream runner* as an extension of streamtubes — an interactively controlled 3D flow visualisation technique that attempts to minimise occlusion, minimise visual complexity, maximise directional cues, and maximise depth cues by letting the user control the length of the streamtubes [62].

Stream polygons

Another extension of streamlines are *stream polygons* used by Schroeder et al. [120]. Stream polygons are tools to visualise 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 visualisation technique used by Brill et al. [9], which visualises 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 [140].

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) [36]. Stream surfaces are very good for texture-based visualisation 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. Stream surfaces present challenges related to occlusion, visual complexity, and interpretation.

Hultquist presents an interactive flow visualisation technique using stream surfaces [35]. Van Wijk presents two follow-up techniques for generating implicit stream surfaces [164]. Cai and Heng [13] address the issues associated with the placement and orientation of stream surfaces in 3D.

Löffelmann et al. present *stream arrows* (see Figure 8, middle) as an enhancement of stream surfaces by separating arrow-shaped portions from a stream surface [76, 75]. Stream arrows address the problem of occlusion associated with 3D flow visualisation, 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 [163]. 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. [161].

Flow volumes

The last (direct) extension of a streamline into 3D described here are *flow volumes* (see Figure 8, 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. [83]. 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 [128].

Becker et al. extend flow volumes to unsteady flow [5]. The resulting unsteady flow volumes are the 3D analogue of streaklines. Considerations are made when extending the visualisation 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.

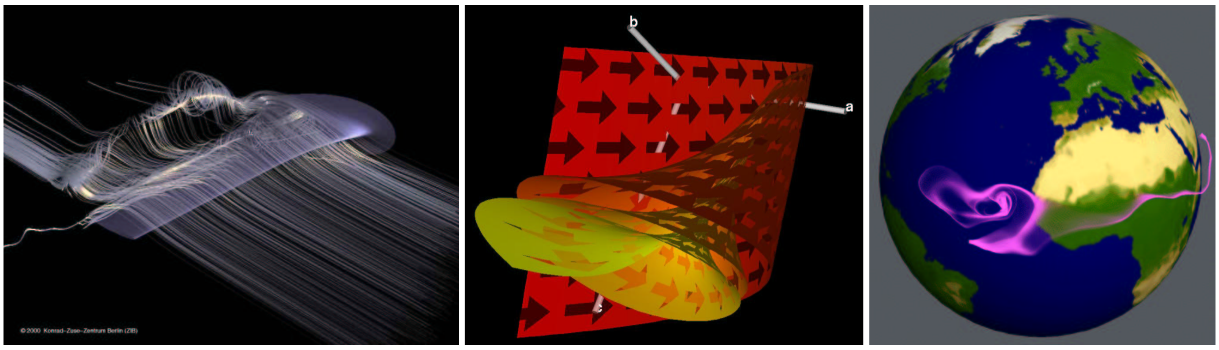


Figure 8: *Examples of flow visualisation using geometric objects — illuminated streamlines (left)* [169], *stream arrows (middle)* [76], *and flow volumes (right)* [83].

5. Feature Extraction

Feature-based flow visualisation is an approach for visualising the flow data at a high level of abstraction. The flow data is described by features, which represent the interesting objects or structures in the data. The original data set is then no longer needed. Because often, only a small percentage of the data is of interest, and the features can be described very compactly, an enormous data reduction can be achieved. This makes it possible to visualise even very large data sets interactively.

The first step in feature-based visualisation is *feature extraction*. The goal of feature extraction is determining, quantifying and describing the features in a data set.

A feature can be loosely defined as any object, structure or region that is of relevance to a particular research problem. In each application, in each data set and for each researcher, a different feature definition could be used. Common examples in fluid dynamics are vortices, shock waves, separation and attachment lines, recirculation zones and boundary layers. In the next section a number of feature-specific detection techniques will be discussed. Although most feature detection techniques are specific for a particular type of feature, in general the techniques can be divided into three approaches: based on image processing, on topological analysis, and on physical characteristics.

5.1. Image Processing

Image processing techniques were originally developed for analysis of 2D and 3D image data, usually represented as scalar (greyscale) values on a regular rectangular grid. The problem of analysing a numerical data set, represented on a grid, is similar to analysing an image data set. Therefore, basic image processing techniques can be used for feature extraction from scientific data. A feature may be distinguished by a typical range of data values, just as different tissue types are segmented from medical images. Edges or boundaries of objects are found by detecting sudden changes in the data values, marked by high gradient magnitudes. Thus, basic image segmentation techniques, such as thresholding, region growing, and edge detection can be used for feature detection. Also, objects may be quantitatively described using techniques such as skeletonisation or principal component analysis. However, a problem is, that in computational fluid dynamics simulations, often grid types are used such as structured curvilinear grids, or unstructured tetrahedral grids. Many techniques from image processing cannot be easily adapted for use with such grids.

5.2. Vector Field Topology

A second approach to feature extraction is the topological analysis of 2D linear vector fields, as introduced by Helman and Hesselink [30, 32], which is based on detection and classification of critical points. These are the points where the vector magnitude is zero. By computing the eigenvalues and eigenvectors of the velocity gradient tensor, the critical points can be classified and tangent curves can be computed. (See Figure 9.) Using this information, a schematic visualisation of the vector field can be generated. (See Figure 15.) Helman and Hesselink have also extended their algorithm to 2D time-dependent and to 3D flows.

Scheuermann et al. presented an algorithm for visualising nonlinear vector field topology [119], because other known algorithms are all based on (piecewise or bi-) linear interpolation, which destroys the topology in case of nonlinear behaviour. Their algorithm makes use of Clifford algebra for computing polynomial approximations in areas with nonlinear local behaviour, especially higher-order singularities.

De Leeuw and Van Liere presented a technique for visualising flow structures using multilevel flow topology [69]. In high-resolution data sets of turbulent flows, the huge number of critical points can easily clutter a flow topology image. The algorithm presented attempts to solve this problem by removing small-scale structures from the topology. This is achieved by applying a pair distance filter which removes

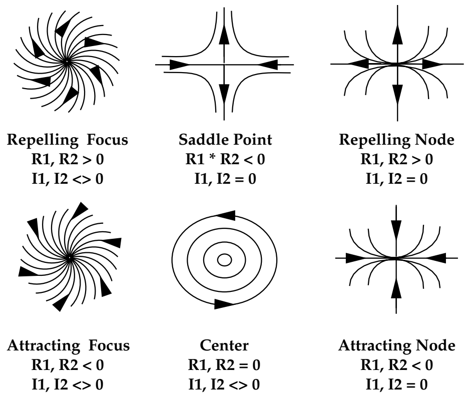


Figure 9: *Vector field topology: critical points classified by the eigenvalues of the Jacobian* [30].

pairs of critical points, that are near each other. This removes small topological structures such as vortices, but does not affect the global topological structure. The threshold distance, which determines which critical points are removed, can be adapted, making it possible to visualise the structure at different levels of detail at different zoom levels.

5.3. Selective Visualisation

A generic approach to feature extraction is Selective Visualisation, which is described by Van Walsum [152]. The feature extraction process is divided into four steps (see Figure 10). The first step is the *segmentation* step. In principle, any segmentation technique can be used, that results in a binary segmentation of the original data set. A very simple segmentation is obtained by thresholding of the original or derived data values; also, multiple thresholds can be combined. The data set resulting from the segmentation step is a binary data set with the same dimensions as the original data set. The binary values in this data set denote whether or not the corresponding points in the original data set are selected. The next step in the feature extraction process is the *clustering* step, in which all points that have been selected are clustered into coherent regions. In the next step, the *attribute calculation* step, these regions are quantified. Attributes, such as

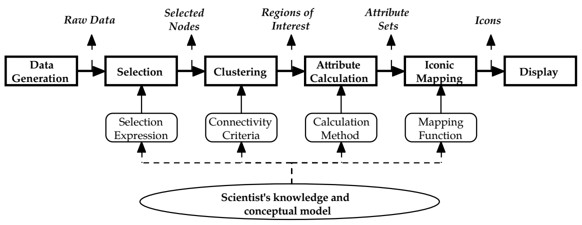


Figure 10: *The feature extraction pipeline* [102].

position, volume and orientation, of the regions are calculated. We now speak of objects, or features, with a number of attributes, instead of clusters of points. Once we have determined these quantified objects, we don't need the original data anymore. With this, we may accomplish a data reduction factor of 1000 or more. In the fourth and final step, *iconic mapping*, the calculated attributes are mapped onto the parameters of certain parametric icons, which are easy to visualise, such as ellipsoids.

6. Feature-based flow visualisation

In this section, a number of feature extraction techniques will be discussed that have been specifically designed for certain types of features. These techniques are often based on physical or mathematical (topological) properties of the flow. Features that often occur in flows are vortices, shock waves and separation and attachment lines.

6.1. Vortex extraction

Features of great importance in flow data sets, both in theoretical and in practical research, are *vortices*. (See Figure 11.) In some cases, vortices (turbulence) have to be impelled, for example to stimulate mixing of fluids, or to reduce drag. In other cases, vortices have to be prevented, for example around aircraft, where they can reduce lift. There

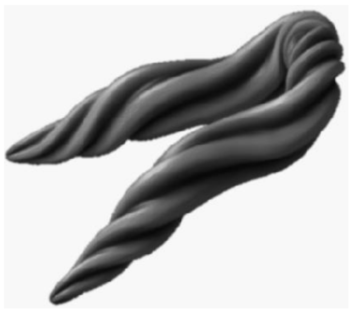


Figure 11: *Reconstruction of a hairpin vortex tube, with grooves indicating velocity* [2].

are many different definitions of vortices and likewise many different vortex detection algorithms. A distinction can be made in algorithms for finding vortex regions and algorithms that only find the vortex cores.

Other overviews of algorithms are given by Roth and Peikert [107] and by Banks and Singer [2].

There are a number of algorithms for finding *regions with vortices*:

- One idea is to find regions with a high vorticity magnitude. Vorticity is the curl of the velocity, that is, $\nabla \times \mathbf{v}$, and represents the local flow rotation, both in speed and direction. However, although a vortex may have a high vorticity magnitude, the reverse is not always true [168]. Villasenor and Vincent present an algorithm for constructing vortex tubes using this idea [151]. They compute the average length of all vorticity vectors contained in small-radius cylinders, and use the cylinder with the maximum average for constructing the vortex tubes.

- Another idea is to make use of helicity instead of vorticity [71, 167]. The helicity of a flow is the projection of the vorticity onto the velocity, that is $(\nabla \times \mathbf{v}) \cdot \mathbf{v}$. This way, the component of the vorticity perpendicular to the velocity is eliminated.
- Another simple idea is to search for regions of low pressure [104].
- Jeong and Hussain define a vortex as a region where two eigenvalues of the symmetric matrix $S^2 + \Omega^2$ are negative, where S and Ω are the symmetric and antisymmetric parts of the Jacobian of the vector field, respectively [41]: $S = \frac{1}{2}(V + V^T)$, and $\Omega = \frac{1}{2}(V - V^T)$. This method is known as the λ_2 method.

The above methods may all work in certain simple flow data sets, but they do not hold, for example, in turbomachinery flows [107], such as shown in Figure 12.

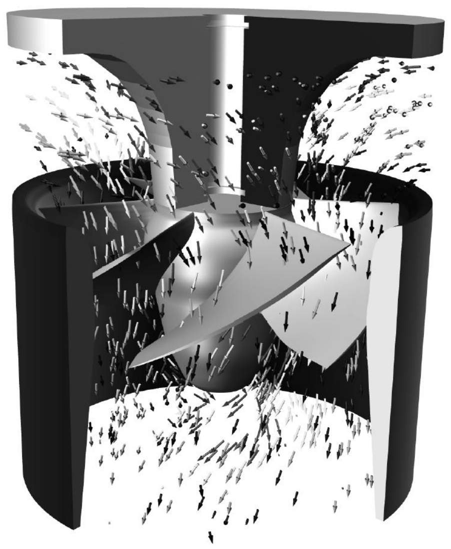


Figure 12: *Visualisation of a turbomachinery flow* [107].

There are also some algorithms specifically for finding *vortex core lines*:

- Globus and Levit presented a method for finding core lines by integrating streamlines from the critical points in the velocity field [25].
- It is also possible to use streamlines of the vorticity field [84], but such an algorithm is very sensitive to the starting location.
- Banks and Singer also use streamlines of the vorticity field, with a correction to the pressure minimum in the plane perpendicular to the vortex core [133, 3].
- Combining the above ideas, Roth and Peikert suggest that a vortex core line can be found where vorticity is parallel to velocity [107]. This sometimes results in coherent structures, but in most data sets it does not give the expected features.
- In the same article, Roth and Peikert suggest that, in linear fields, the vortex core line is located where the Jacobian has one real-valued eigenvector, and this eigenvector is parallel to the flow [107]. However, in their own application of turbomachinery flows, the assumption of a linear flow is too simple. The same algorithm is presented by Sujudi and Haimes [136].
- Recently, Jiang et al. presented a new algorithm for vortex core region detection [42], which is based on ideas derived from combinatorial topology. The algorithm determines for each cell if it belongs to the vortex core, by examining its neighbouring vectors.

A few of these algorithms will be reviewed in detail, next.

Banks and Singer developed a predictor-corrector algorithm for finding vortex cores [3]. After initialisation, vortex cores are tracked by predicting in the direction of the vorticity vector and correcting to the pressure minimum in the plane perpendicular to that vorticity vector. Next, they create vortex tubes, by computing cross sections of the vortices, in a plane perpendicular to the vortex core. They use a threshold of the pressure as a selection criterion, in combination with the restriction that the angle between the vorticity vector at any point on the cross section and the vorticity vector at the vortex core is no more than ninety degrees.

Sujudi and Haimes developed an algorithm for finding the centre of swirling flow in 3D vector fields and implemented this algorithm in pV3 [136]. Although pV3 can use many types of grids, the algorithm has been implemented for tetrahedral cells. When using data sets with other types of cells, these first have to be decomposed into tetrahedral cells. This is done for efficiency, because linear interpolation for the velocity can be used in the case of tetrahedral cells. The algorithm is based on critical-point theory and uses the eigenvalues and eigenvectors of the velocity gradient tensor or rate-of-deformation tensor. The algorithm works on each point in the data set separately, making it very suitable for parallel processing. The algorithm searches for points where the velocity gradient tensor has one real and two complex-conjugate eigenvalues and the velocity is in the direction of the eigenvector, corresponding to the real eigenvalue. The algorithm results in large coherent structures when a strong swirling flow is present, and the grid cells are not too large. The algorithm is sensitive to the strength of the swirling flow, resulting in incoherent structures or even no structures at all in weak swirling flows. Also, if the grid cells are large, or irregularly sized, the algorithm has difficulties finding coherent structures or any structures at all.

Kenwright and Haimes also studied the eigenvector method and concluded that it has proven to be effective in many applications [53]. The drawbacks of the algorithm are that it does not produce contiguous lines. Line segments are drawn for each tetrahedral element, but they are not necessarily continuous across element boundaries. Furthermore, when the elements are not tetrahedra, they have to be decomposed into tetrahedra first, introducing a piecewise linear approximation for a nonlinear function. Another problem is that flow features are found that are not vortices. Instead, swirling flow is detected, of which vortices are an example. However, swirling flow also occurs in the formation of boundary layers. Finally, the eigenvector method is sensitive to other nonlocal vector features. For example, if two axes of swirl exist, the algorithm will indicate a rotation that is a combination of the two swirl directions. The eigenvector method has successfully been integrated into a finite element solver for guiding mesh refinement around the vortex core [16].

Roth and Peikert have developed a method for finding core lines using higher-order derivatives, making it possible to find strongly curved or bent vortices [108]. They observe that the eigenvector method is equivalent to finding points where the acceleration **a** is parallel to the velocity **v**, or equivalently, to finding points of zero curvature. The acceleration **a** is defined as:

$$\mathbf{a} = \frac{D\mathbf{v}}{Dt}, \tag{6}$$

where the notation $\frac{Df}{Dt}$ is used for the *derivative following a particle*, which is defined, in a steady flow, as $\nabla f \cdot \mathbf{v}$. Therefore:

$$\mathbf{a} = \frac{D\mathbf{v}}{Dt} = \nabla\mathbf{v} \cdot \mathbf{v} = J \cdot \mathbf{v}, \tag{7}$$

with J the Jacobian of **v**, that is the matrix of its first derivatives.

Roth and Peikert improve the algorithm by defining vortex cores as points where

$$\mathbf{b} = \frac{D\mathbf{a}}{Dt} = \frac{D^2\mathbf{v}}{Dt^2} \tag{8}$$

is parallel to **v**, that is, points of zero torsion. The method involves computing a higher-order derivative, introducing problems with accuracy, but it performs very well. In comparison with the eigenvector method, this algorithm finds strongly curved vortices much more accurately. Roth and Peikert also introduce two attributes for the core lines: the strength of rotation and the quality of the solution. This makes it possible for the user to impose a threshold on the vortices, to eliminate weak or short vortices. Peikert and Roth have also introduced a new operator, the "parallel vectors" operator [93], with which they are able to mathematically describe a number of previously developed methods under one common denominator. Using this operator they can describe methods based on zero curvature, ridge and valley lines, extremum lines and more.

Jiang et al. recently presented a new approach for detecting vortex core regions [42]. The algorithm is based on an idea which has been derived from Sperner's lemma in combinatorial topology, which states that it is possible to deduce the properties of a triangulation, based on the information given at the boundary vertices. The algorithm uses this fact to classify points as belonging to a vortex core, based on the vector orientation at the neighbouring points. In 2D, the algorithm is very simple and straightforward, and has only linear complexity. In 3D, the algorithm is somewhat more difficult, because it first involves computing the vortex core direction, and next, the 2D algorithm is applied to the velocity vectors projected onto the plane perpendicular to the vortex core direction. Still, also the 3D algorithm has only linear complexity.

The above described methods all use a local criterion for determining on a point-to-point basis where the vortices are located. The next algorithms use global, geometric criteria for determining the location of the vortices. This is a consequence of using another vortex definition.

Sadarjoen and Post present two geometric methods for extracting vortices in 2D fields [112]. The first is the curvature centre method. For each sample point, the algorithm computes the curvature centre. In the case of vortices, this would result in a high density of centre points near the centre of the vortex. The method works but has the same limitations as traditional point-based methods, with some false and some missing centres. The second method is the winding-angle method, which has been inspired by the work of Portela [94]. The method detects vortices by selecting and clustering looping streamlines. The winding angle α_w of a streamline is defined as the sum of the angles between the different streamline segments. Streamlines are selected that have made at least one complete rotation, that is, $\alpha_w \geq 2\pi$. A second criterion checks that the distance between the starting and ending points is relatively small. The selected streamlines are used for vortex attribute calculation. The geometric mean is computed of all points of all streamlines belonging to the same vortex. An ellipse fitting is computed for each vortex, resulting in an approximate size and orientation for each vortex. Furthermore, the angular velocity and rotational direction can be computed. All these attributes can be used for visualising the vortices. (See Figure 13.)

6.2. Shock wave extraction

Shock waves are also important features in flow data sets, and can occur, for example, in flows around aircraft. (See Figure 14.) Shock waves could increase drag and cause structural failure, and therefore, are important phenomena to study. Shock waves are characterised by discontinuities in physical flow quantities such as pressure, density and velocity. Therefore, shock detection is comparable to edge detection, and similar principles could be used as in image processing. However, in numerical simulations, the discon-

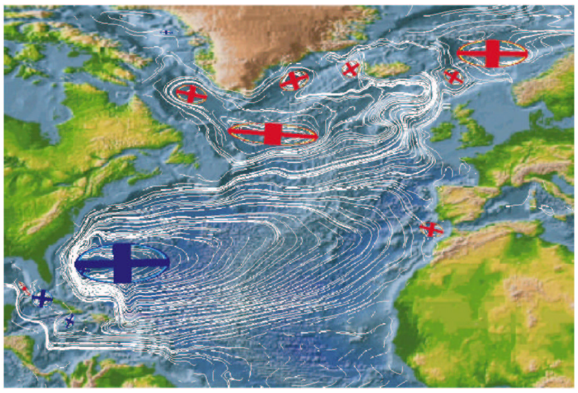


Figure 13: *Flow in the Atlantic Ocean, with streamlines and ellipses indicating vortices. Blue and red ellipses indicate vortices rotating clockwise and counterclockwise, respectively* [113].

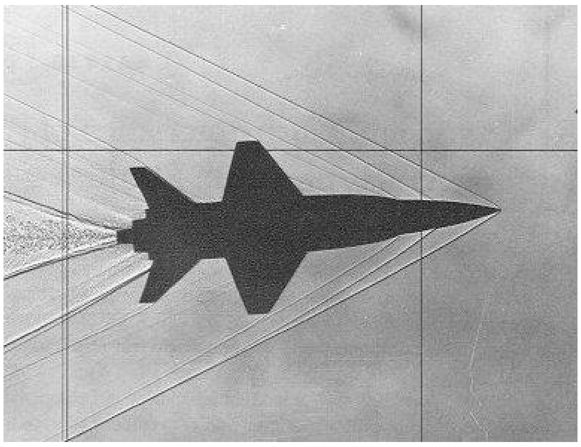


Figure 14: *Shock waves around a model of a X-15 in a wind tunnel with an airflow at Mach 3.5. Image from the NASA website.*

tinuities are often smeared over several grid points, due to the limited resolution of the grid. Ma et al. have investigated a number of techniques for detecting and for visualising shock waves [79]. Detecting shocks in two dimensions has been extensively investigated [64, 86, 105]. However, these techniques are in general not applicable to shocks in three dimensions. They also describe a number of approaches for visualising shock waves. The approach of Haimes and Darmofal [28] is to create isosurfaces of the Mach number normal to the shock, using a combined density gradient/Mach number computation. Van Rosendale presents a two-dimensional shock-fitting algorithm for unstructured grids [105]. The idea relies on the comparison of density gradients between grid nodes.

Ma et al. compare a number of algorithms for shock extraction and also present their own technique [79]:

- The first idea is to create an isosurface of the points where the Mach number is one. However, this results in the sonic surface, which, in general, does not represent a shock.
- Theoretically, a better idea is to create an isosurface of the points where the normal Mach number is equal to one. However, if the surface is unknown, it is impossible to compute the Mach number, normal to the surface.
- This problem can be resolved, by approximating the shock normal with the density gradient, since a shock is also associated with a large gradient of the density. Therefore, $\nabla\rho$ is (roughly) normal to the shock surface. Thus, the algorithm computes the Mach number in the direction of, or projected onto, the density gradient. The shock surface is constructed from the points where this Mach number equals one. This algorithm is also used by Lovely and Haimes [78], but they define the shock region as the region within the isosurface of Mach number one, and use filtering techniques to reconstruct a sharp surface.
- Pagendarm presented an algorithm that searches for maxima in the density gradient [91]. The first and second derivatives of the density in the direction of the velocity are computed. Next, zero-level isosurfaces are constructed of the second derivative, to find the extrema in the density gradient. Finally, the first derivative is used to select only the maxima, which correspond to shock waves, and discard the minima, which represent expansion waves. This can be done by selecting only positive values of the first derivative. However, the second derivative can also be zero in smooth regions with few disturbances. In these regions the first derivative will be small, therefore, these regions can be excluded by discarding all points where the first derivative is below a certain threshold ε. Of course, this poses the problem of finding the correct ε. When the value is too small, erroneous shocks will be found, but if the value is too large, parts of the shocks could disappear. This algorithm can also be used for finding discontinuities in other types of scalar fields, and thus for finding other types of features.
- Ma et al. present an adapted version of this algorithm, which uses the normal Mach number to do the selection in the third step [79]. Again, in the first and second step, the zero-level isosurfaces of the second directional derivative of the density are constructed. But for discriminating shock waves from expansion waves and smooth regions, the normal Mach number is used. More precisely, those points are selected where the normal Mach number is close to one. Here also, a suitable neighbourhood of one has to be chosen.

6.3. Separation and attachment line extraction

Other features in flow data sets are separation and attachment lines on the boundaries of bodies in the flow. These are the lines where the flow abruptly moves away from or returns to the surface of the body. These are important features in aerodynamic design because they can cause increased drag and reduced lift [106], and therefore, their occurrence should be prevented or at least minimised. Helman and Hesselink use

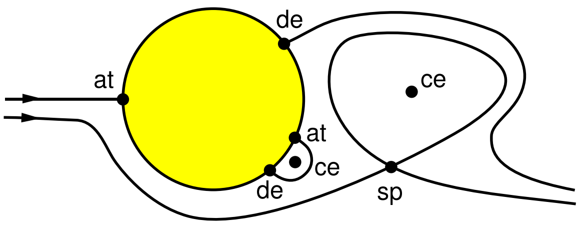


Figure 15: *Vector field topology: a topological skeleton of a flow around a cylinder* [32].

vector field topology to visualise flow fields [31, 32]. In addition to the critical points, the attachment and detachment nodes on the surface of a body determine the topology of the flow. (See Figure 15.) The attachment and detachment nodes are not characterised by a zero velocity, because they only occur in flows with a no-slip condition, that is, all points on the boundaries of objects are constrained to have zero velocity. Instead, they are characterised by a zero tangential velocity. Therefore, streamlines impinging on the surface terminate at the attachment or detachment node, instead of being deflected along the surface.

Globus et al. designed and implemented a system for analysing and visualising the topology of a flow field with icons for the critical points and integral curves starting close to the critical points [25]. The system is also able to visualise attachment and detachment surfaces and vortex cores.

Pagendarm and Walter [92] and De Leeuw et al. [66] used skin-friction lines for visualising attachment and detachment lines in the blunt fin data set. For visualising these lines, the wall shear vector τ_w is computed, which is the gradient of the velocity magnitude $|\mathbf{v}|$, projected along the normal onto the wall:

$$\tau_w = \nabla|\mathbf{v}| - (\nabla|\mathbf{v}| \cdot \mathbf{n})\mathbf{n}, \qquad (9)$$

where $\mathbf{n}$ is the unit vector normal to the wall. Next, a standard streamline algorithm is used to integrate the skin-friction lines from the shear vector field. These skin-friction lines show the location of separation and attachment of the flow at the wall. (See Figure 16.)

Kenwright gives an overview of existing techniques for visualising separation and attachment lines and presents a new automatic feature detection technique for locating these lines, based on concepts from 2D phase plane analysis [52]. Some common approaches are:

- Particle seeding and computation of integral curves, such as streamlines and streaklines, which are constrained to the surface of the body. These curves merge along separation lines.
- Skin-friction lines can be used, analogous to surface oil flow techniques from wind tunnel experiments [92]. (See above.)
- Texture synthesis techniques can be used to create continuous flow patterns rather than discrete lines [66].
- Helman and Hesselink can automatically generate separation and attachment lines from their vector field topology [31]. These lines are generated by integrating curves from the saddle and node type critical points in the direction of the real eigenvector. However, only closed separations are found, that is, the curves start and end at critical points.

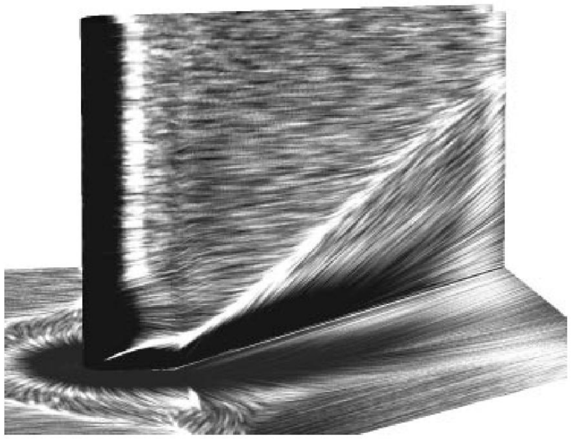


Figure 16: *Skin-friction on a blunt fin from a flow simulation at Mach 5, visualised with spot noise* [66].

Open separation does not require separation lines to start or end at critical points, and is therefore not detected by flow topology. Open separation has been observed in experiments, but had not previously been studied in flow simulations. However, the algorithm presented by Kenwright does detect both closed and open separation lines. The theory for this algorithm is based on concepts from linear phase plane analysis. It is assumed that the computational domain on the surface can be subdivided into triangles and the vector components are given at the vertices. The algorithm is executed for each triangle, making it suitable for parallelisation. For each triangle, a linear vector field is constructed satisfying the vectors at the vertices. If the determinant of the Jacobian matrix is nonzero, the algorithm continues by calculating the eigenvalues and eigenvectors of the Jacobian. Every triangle has a critical point somewhere in its vector field. The linear vector field is translated to this critical point and the coordinate system is changed so that the eigenvectors are orthogonal. This (x,y) plane is also referred to as the Poincaré phase plane. By computing tangent curves in the phase plane, we obtain the phase portrait of the system. For a saddle, the tangent curves or streamlines converge along the x and y axes. For a repelling node, they converge along the x axis and for an attracting node, they converge along the y axis. If the phase portrait is a saddle or a repelling node, the intersection of the x axis with the triangle is computed. If it intersects, the line segment will form part of an attachment line. If the phase portrait is a saddle or an attracting node, the intersection of the y axis with the triangle is computed, and if it does intersect, the line segment will form part of a separation line.

A problem with this algorithm is that disjointed line segments are computed instead of continuous attachment and separation lines. Other problems occur when the flow separation or attachment is relatively weak, or when the assumption of locally linear flow is not correct.

Kenwright et al. present two algorithms for detecting separation and attachment lines [54]. The first is the algorithm discussed above, the second is the parallel vector algorithm. Both algorithms use eigenvector analysis of the velocity gradient tensor. However, the first is element-based and results in disjointed line segments, while the second is point-based and will result in continuous lines.

In the parallel vector algorithm, points are located where one of the eigenvectors $\mathbf{e}_i$ of the gradient $\nabla\mathbf{v}$ is parallel to the vector field $\mathbf{v}$, that is, points where the streamline curvature is zero, or in formula:

$$\mathbf{e}_i \times \mathbf{v} = \mathbf{0}. \tag{10}$$

The velocity vectors and the eigenvectors can be determined at the vertices of the grid and interpolated within the elements. At the vertices, $\mathbf{e}_i \times \mathbf{v}$ is calculated for both eigenvectors, but only if both eigenvectors are real, that is, the classification of $\nabla\mathbf{v}$ at the vertex is either a saddle or a node. If the cross product $\mathbf{e}_i \times \mathbf{v}$ changes sign across an edge, that means an attachment or separation line intersects the edge. The intersection point can then be found by interpolation along the edge. The attachment and separation lines can be constructed by connecting the intersection points in each element. The distinction between attachment and separation can be made easily, because attachment will occur where $\mathbf{v}$ is parallel to the smallest $\mathbf{e}_i$ and separation where $\mathbf{v}$ is parallel to the largest $\mathbf{e}_i$. Another set of lines is detected with this algorithm, the inflection lines. These can easily be filtered out by checking if:

$$\nabla(\mathbf{e}_i \times \mathbf{v}) \cdot \mathbf{v} = \mathbf{0}. \tag{11}$$

This will not be true for inflection lines.

Both algorithms discussed by Kenwright et al. correctly identify many separation and attachment lines, but may fail in identifying curved separation lines [54]. The parallel vector algorithm will result in continuous lines, whereas the phase plane algorithm results in discontinuous line segments. Both algorithms do detect open separation lines, which do not start or end at critical points.

6.4. Other types of features

There are other types of features, such as recirculation zones and boundary layers. Work has been done in extracting these features, for example by Haimes [27] and by Sadarjoen and Post [110]. Hunt et al. give quantitative criteria for dividing a flow into three areas, with specific characteristics: eddies, streams, and convergence zones [37].

7. Feature tracking and event detection

In time-dependent data sets, features are objects that evolve in time. Determining the correspondence between features in successive time steps, that actually represent the same object at different times, is called the *correspondence problem*. Feature tracking is involved with solving this correspondence problem. The goal of feature tracking is to be able to describe the evolution of features through time. During the evolution, certain *events* can occur, such as the interaction of two or more features, or significant shape changes of features. Event detection is the process of detecting such events, in order to describe the evolution of the features even more accurately.

There are two basic approaches to solving the correspondence problem. The first is based on region correspondence, the second on attribute correspondence.

7.1. Region correspondence

Region correspondence involves comparing the regions of interest obtained by feature extraction. Basically, the binary images from successive time steps, containing the features found in these time steps, are compared on a cell-to-cell basis. Correspondence can be found using a minimum distance or a maximum cross-correlation criterion [26] or by minimising an affine transformation matrix [50]. It is also possible to extract isosurfaces from the four-dimensional time-dependent data set [159], where time is the fourth dimension. The correspondence is then implicitly determined by spatial overlap between successive time steps. This criterion is simple, but not always correct, as objects can overlap but not correspond, or correspond but not overlap. Silver and Wang explicitly use the criterion of spatial overlap instead of creating isosurfaces in four dimensions [130, 131, 132]. They prevent correspondence by accidental overlap, by checking the volume of the corresponding features and taking the best match. This is also the idea of attribute correspondence, which is discussed next. By using spatial overlap, certain events are implicitly detected, such as a *bifurcation* when a feature in one time step overlaps with two features in the next time step. Event detection is also discussed more elaborately later, in Section 7.3.

7.2. Attribute correspondence

With attribute correspondence, the comparison of features from successive frames is performed on the basis of the attributes of the features, such as the position, size, volume, and orientation. These attributes can be computed in the feature extraction phase, (see Section 5.3,) and can be used for description and for visualisation of the features, and also for feature tracking, as described here. The original grid data is not needed anymore. Samtaney et al. use the attribute values together with user-provided tolerances to create correspondence criteria [115]. For example, for position the following could be used:

$$dist(pos(O_{i+1}), pos(O_i)) \leq T_{dist}, \tag{12}$$

where $pos(O_i)$ and $pos(O_{i+1})$ are the positions of the objects in time steps i and $i+1$, respectively, and T_{dist} is the user-provided tolerance. For scalar attributes, the difference or the relative difference could be used. For example, to test the relative difference of the volume, the following formula can be used:

$$\frac{vol(O_{i+1}) - vol(O_i)}{\max(vol(O_{i+1}), vol(O_i))} \leq T_{vol}, \tag{13}$$

where $vol(O_i)$ and $vol(O_{i+1})$ are the volumes of the features in the two time steps, and T_{vol} is the tolerance given by the user. Events such as a bifurcation can also be tested. If a feature in time step i splits into two features in time step $i+1$, the total volume after the event has to be approximately the same as before the event. The same formula can be used as for the normal volume test, except that $vol(O_{i+1})$ in this case equals the sum of the volumes of the separate features. The position criterion in case of a bifurcation event could involve the weighted average of the individual positions after the event, where the positions are weighted with the volume:

$$dist(pos(O_i), \frac{\sum(vol(O_{i+1}) \cdot pos(O_{i+1}))}{\sum(vol(O_{i+1}))}) \leq T_{dist}, \tag{14}$$

where O_{i+1} now represents all objects in time step $i+1$ that are involved in the event.

Sethi et al. present a method for image-based motion analysis, with the use of markers or tokens [123]. The basic concept is *smoothness of motion* of feature point *trajectories* in *property space*. Properties or attributes of features are represented by points in property space. These points move through the property space over time, and the algorithm tries to find the smoothest paths or trajectories in this property space. The notion of *property coherence* is used, that is, the properties are supposed to change gradually. Two algorithms are described to find the smoothest trajectories: Modified Greedy Exchange and Simulated Annealing. We will describe the former here. The basic idea of both algorithms is to create initial trajectories by connecting the closest points in property space, and then to iteratively refine the trajectories to maximise the total smoothness of all trajectories. In the Greedy Exchange algorithm, this optimisation is done by exchanging tokens between trajectories and computing the gain in smoothness. The exchange with the maximum gain is chosen. The process is repeated, both forward and backward in time, until no more exchanges are made. The property coherence is a measure for three consecutive points in property space. When we call these points, from three consecutive time steps, a, b and c, the property coherence is defined as:

$$\begin{aligned} F(a,b,c) = \; & w_1 \left(1 - \frac{\vec{ab}\cdot\vec{bc}}{\|\vec{ab}\|\|\vec{bc}\|}\right) + \\ & w_2 \left(1 - 2\frac{\sqrt{\|\vec{ab}\|\|\vec{bc}\|}}{\|\vec{ab}\|+\|\vec{bc}\|}\right) \end{aligned} \tag{15}$$

The first term on the right hand side gives a measure for the change in direction between the vectors $\vec{ab}$ and $\vec{bc}$, and the second term gives a measure for the change in length of these vectors. Both these measures are combined with user-provided weights w_1 and w_2. Because different properties can have different characteristics, the axes in the property space can be scaled using suitable scaling factors. The normal formulas for computing the inner product of two vectors and the length of a vector are therefore adapted as follows:

$$\begin{aligned} \vec{ab}\cdot\vec{bc} &= \sum_{i=1}^{k} s_i(b_i - a_i)(c_i - b_i) \\ \|\vec{ab}\| &= \sqrt{\sum_{i=1}^{k} s_i(b_i - a_i)^2}, \end{aligned} \tag{16}$$

where s_i is the scaling factor for the i^{th} axis, with $\sum s_i = 1$, and a_i is the i^{th} component of the k-dimensional property vector a.

Reinders et al. describe an algorithm for feature tracking, that is based on prediction and verification [99, 100]. This algorithm is based on the assumption that features evolve predictably. That means, if a part of the evolution of a feature (*path*) has been found, a prediction can be made into the next time step (*frame*). Then, in that next time step, a feature is sought, that corresponds to the prediction. If a feature is found that matches the prediction within certain user-provided tolerances, the feature is added to the evolution and the search is continued to the next time step. When no more features can be added to the path, a new path is started. In this manner, all frames are searched for starting points, both in forward and backward time direction, until no more paths can be created.

A path is started by trying all possible combinations of features from two consecutive frames and computing the prediction to the next frame. Then, the prediction is compared to the candidate features in that frame. If there is a match between the prediction and the candidate, a path is started. To avoid any erroneous or coincidental paths, there is a parameter for the minimal path length, which is usually set to 4 or 5 frames. A candidate feature can be defined in two ways. All features in the frame can be used as candidates, or only unmatched features can be used, that is, those features that have not yet been assigned to any path. The first definition ensures that all possible combinations are tested and that the best correspondence is chosen. However, it could also result in features being added to more than one path. This has to be resolved afterwards. Using the second definition is much more efficient, because the more paths are found, the less unmatched features have to be tested. However, in this case, the results depend on the order in which the features are tested. This problem can be solved by starting the tracking process with strict tolerances and relaxing the tolerances in subsequent passes.

The prediction of a feature is constructed by linear extrapolation of the attributes of the features from the last two frames. Other prediction schemes could also be used, for example, if a-priori knowledge of the flow is available.

The prediction is matched against real features using correspondence criteria, similar to the ones used by Samtaney et al. as discussed above [115]. For each attribute of the features, a correspondence function can be created, which returns a positive value for a correspondence within the given tolerance, with a value of 1 for an exact match, and a negative value for no correspondence. Each correspondence function is assigned a weight, besides the tolerance. Using this weight, a weighted average is calculated of all correspondence functions, resulting in the correspondence factor between the two features. For this correspondence factor, the same applies as for the separate correspondence functions, that is, a positive value indicates a correspondence, with 1 indicating a perfect match. A negative correspondence factor means no match.

7.3. Event detection

After feature tracking has been performed, event detection is the next step. Events are the temporal counterparts of spatial features in the evolution of features. For example, if the path or evolution of a feature ends, it can be interesting to determine why that happens. It could be that the feature shrinks and vanishes, or that the feature moves to the boundary of the data set and disappears, or that the feature merges with another feature and the two continue as one. Samtaney et al. introduced the following events: continuation, creation, dissipation, bifurcation, amalgamation [115]. (See Figure 17.) Reinders et al. developed a feature tracking system that is able to detect these and other events [100]. The terminology they use is *birth* and *death* instead of creation and dissipation, and *split* and *merge* for bifurcation and amalgamation. Furthermore, they can detect *entry* and *exit* events, where a feature moves beyond the boundary of the data set. Finally, for a specific, graph-type feature, the system is able to detect changes in topology. It discriminates *loop* and *junction* events. (See Figure 18.) Many other types of events can be envisioned, but for each type specific detection criteria have to be provided.

For event detection, just as for feature tracking, only the feature attributes are used. Analogous to the correspondence functions, for event detection, event functions are computed.

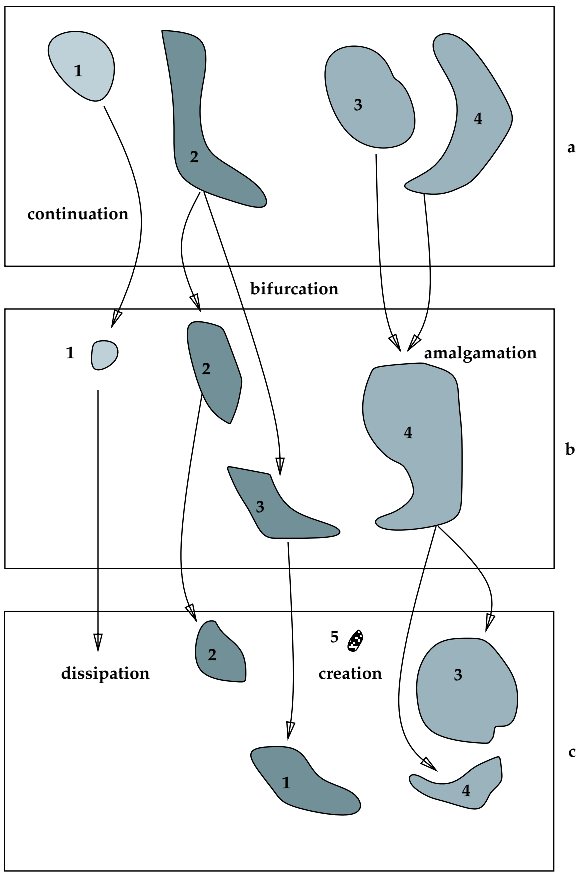


Figure 17: *The different types of events as introduced by Samtaney et al.* [115].

For example, to detect a death event, two conditions must hold. First, the volume of the feature must decrease. And second, the volume of the prediction must be very small or negative. The event function for this event returns a positive value if the volume of the prediction is within the user-provided tolerance, and 1 if the volume of the prediction is negative. If the volume is not within the tolerance, the returned value will be negative. The event functions for the separate attributes are combined into a single factor, which determines if the event is a death event. A birth event can be detected by doing the same tests in the backward time direction.

Similarly, the tests for split and merge events, and for entry and exit events are each other's reverse in time.

8. Visualisation of features and events

The final step in the feature extraction process is, of course, the visualisation of the features. A number of techniques will be covered in this section. The most straightforward visualisation is to show the nodes in the data set, that have been

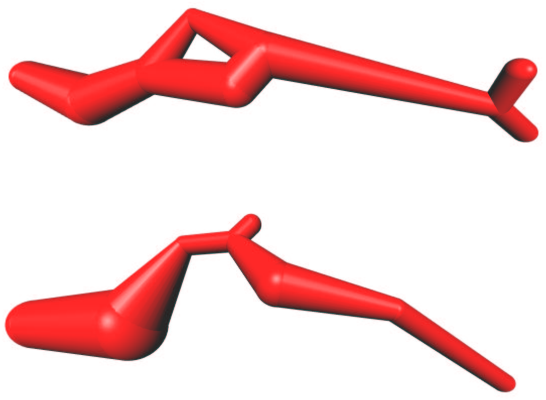


Figure 18: *A loop event has occurred. In the top figure, the feature contains a loop, in the bottom figure, the next frame, the loop has disappeared* [97].

selected in the first step of the feature extraction pipeline, the selection. This step results in a binary data set, with each value indicating whether the corresponding node has been selected or not. This binary data set can be visualised, for example, with crosses at the selected nodes. In Figure 19,

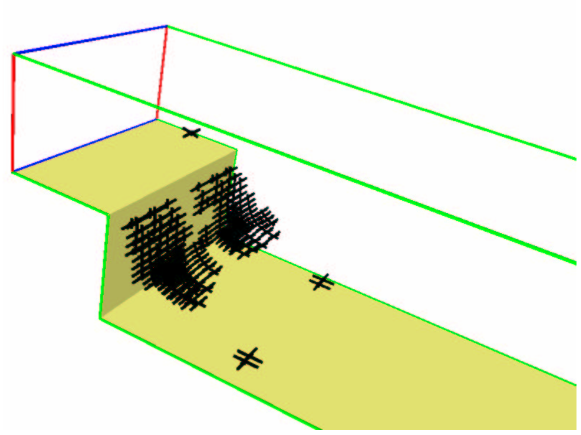


Figure 19: *Visualisation of the selected points in the backward-facing-step data set* [114].

such a visualisation is shown. The visualisation is of a simulation of the flow behind a backward-facing step. The feature that is visualised here is a recirculation zone, behind the step. The points were selected with the criterion: normalised helicity $H > 0.6$.

Another simple visualisation technique is to use isosurfaces. This can be done on the binary data set, resulting from the selection step, or, if the selection expression is a simple threshold, directly on the original data set. This results in isosurfaces enclosing the selected regions.

Also, other standard visualisation techniques can be used in combination with the boolean data set resulting from the selection step. For example, in a 3D flow data set, using the standard methods for seeding streamlines or streamtubes, will not provide much information about the features and will possibly result in visual clutter. However, if the selected points are used to seed streamlines, both backward and forward in time, this can provide useful information about the features and their origination. See Figure 20, for an example,

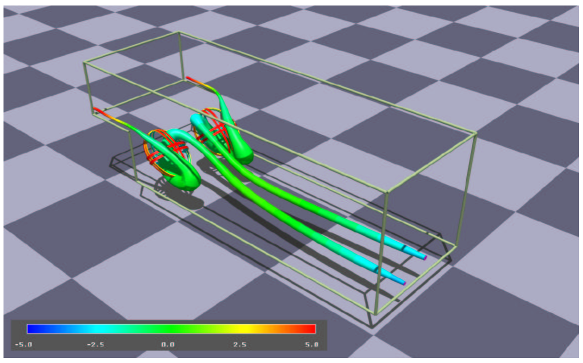


Figure 20: *Visualisation with streamtubes of the recirculation in the backward-facing-step data set* [153].

where two streamtubes are shown in the backward-facing-step data set. The radius of the tubes is inversely proportional to the square root of the local velocity magnitude, and the colour of the tubes corresponds to the pressure.

If, instead of the separate selected points, the attributes are used, that have been computed in the feature extraction process, then parametric icons can be used for visualising the features.

If an ellipsoid fitting of the selected clusters has been computed, there are three attribute vectors: the centre position, the axis lengths, and the axis orientations, which can be mapped onto the parameters of an ellipsoid icon. This is a simple icon, but very efficient and accurate. It can be represented with 9 floating-point values, and is therefore space-efficient. Furthermore, it can be very quickly visualised, and although it is simple, it gives an accurate indication of the position and volume of a feature. In Figure 21, an ellipsoid fitting is computed from the selected points in Figure 19. In Figure 22, vortices are shown from a CFD simulation with turbulent vortex structures. The features have been selected by a threshold on vorticity magnitude. They are being visualised with isosurfaces and ellipsoids. It is clearly visible that, in this application, with the strongly curved features, the ellipsoids do not give a good indication of the shape of the features. But, as mentioned above, the position and volume attributes of the ellipsoids will be accurate, and can be

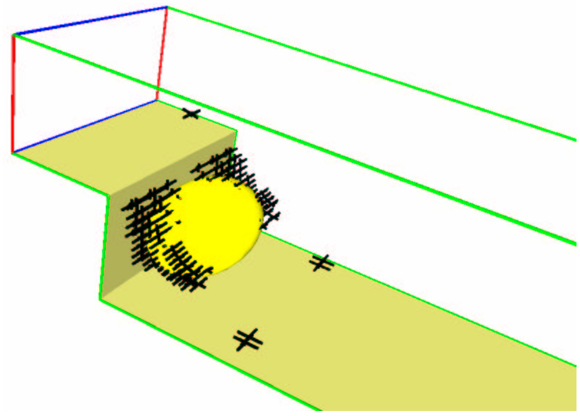


Figure 21: *An ellipsoid fitting computed from the selected points in the backward-facing-step data set* [114].

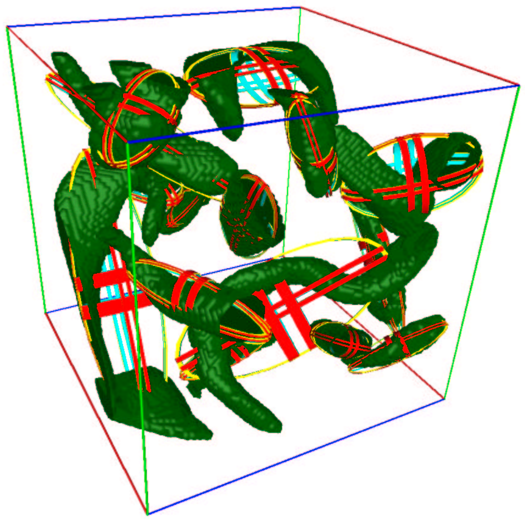


Figure 22: *Vortices in a data set with turbulent vortex structures, visualised using isosurfaces and ellipsoids* [97].

used for feature tracking. In Figure 23, the flow past a tapered cylinder is shown. Streamlines indicate the flow direction, and rotating streamlines indicate vortices. The vortices are selected by locating these rotating streamlines, using the winding-angle method [112]. Ellipsoids are used to visualise the vortices, with the colour indicating the rotational direction. Green means clockwise rotation, red means counterclockwise rotation. The slice is coloured with λ_2, which is the second-largest eigenvalue of the tensor $S^2 + \Omega^2$. (See Section 6.1.)

The tapered cylinder data set consists of a number of horizontal slices, such as the one in Figure 23, but the vortices are naturally three-dimensional structures. Reinders

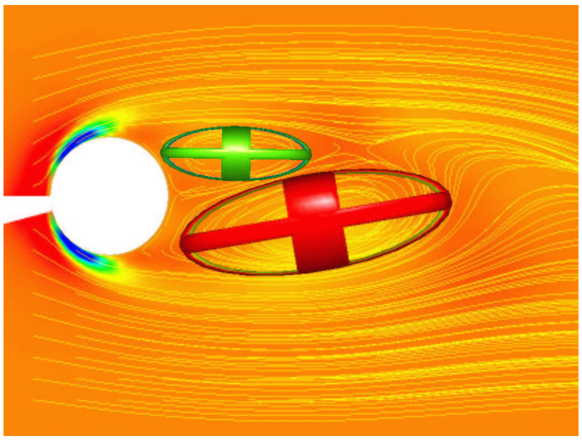


Figure 23: *Vortices behind a tapered cylinder. The colour of the ellipsoids represents the rotational direction* [112].

et al. created these vortex structures by performing feature tracking in a spatial dimension instead of tracking in time [101]. First, feature extraction was performed in the two-dimensional slices. This resulted in two-dimensional vortices, which were represented by a special type of ellipse icon. Next, these 2D features were tracked in the z-direction, forming 3D vortex structures. Figure 24 shows an image of the resulting features. The 2D icons are ellipses with a number of curved spokes. The curvature of the spokes indicates the rotational direction of the vortices, and the number of spokes represents the rotational speed. The 3D icons are constructed by connecting the centre points of the 2D ellipses.

For the 3D vortices in Figure 22, an other type of icon has to be used, if we want to visualise the strongly curved shape of the features. Reinders et al. present the use of skeleton graph descriptions for features, with which they can create icons that accurately describe the topology of the features, and approximately describe the shape of the features [98]. Compare the use of ellipsoid icons with the use of skeleton icons in Figure 25.

For visualising the results of feature tracking, it is of course essential to visualise the time dimension. The most obvious way is to animate the features, and to give the user the opportunity to browse through the time steps, both backward and forward in time. Figure 26 shows the player from the feature tracking program, developed by Reinders [100]. On the left of the figure, the graph viewer is shown, which gives an abstract overview of the entire data set, with the time steps on the horizontal axis, and the features represented by nodes, on the vertical axis. The correspondences between features from consecutive frames are represented by edges in the graph, and therefore, the evolution of a feature in time, is represented by a path in the graph. On the right of the figure, the feature viewer is shown, in which the feature icons from the current frame are displayed. Also, a control panel

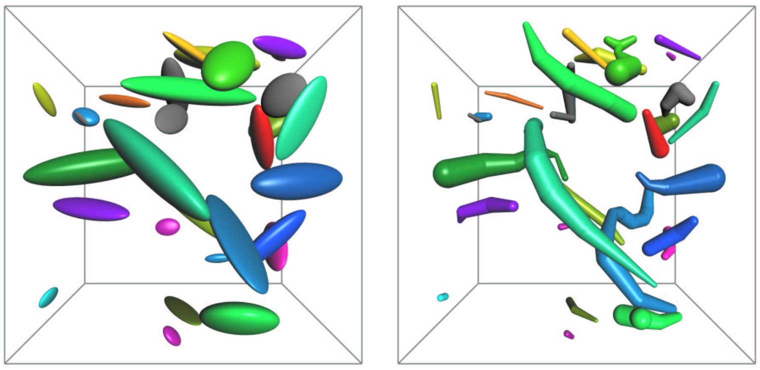


Figure 25: *Turbulent vortex structures represented by ellipsoid icons (left) and skeleton icons (right)* [97].

is visible, with which the animation can be started, paused, and played forward and backward.

The graph viewer can also be used for visualising events [97]. For each event, a specific icon has been created, which is mapped onto the nodes of the graph, so that the user can quickly see which events occur where, and how often they occur. In Figure 27, the graph viewer is shown, with a part of the graph, containing a number of events. Each event is clearly recognisable by its icon. In Figure 28, two frames are shown, between which a split event has occurred. In both frames, the features are shown with both ellipsoid and skeleton icons. The advantage of the use of skeleton icons in this application is obvious. Because the shape of the features is much more accurately represented by the skeleton icons, changes in shape and events such as these are much more easily detected.

9. Conclusions and future prospects

A state-of-the-art report must end with an assessment: what has been achieved in flow visualisation during the last 15 years? Have the problems been solved? Are the results applied in practice? What are the remaining challenges?

A large number of techniques has been developed and refined. In general, which techniques are the best, depends strongly on the purpose of the visualisation: the research problems addressed, the methods and approaches used, and the personal interest of the researcher or engineer. Users may also have different purposes, such as exploration, detailed analysis, or presentation. Therefore, we believe that a large variety of techniques must be available to allow researchers to choose the most suitable technique for their purpose. In this sense, good progress has been made.

A very successful group is the texture-based techniques (see Section 3), mainly used for 2D flows and surface flow fields. They are very suitable for animation, both of stationary and time-dependent flows. Performance limitations seem to be overcome [165], and interactive use with unsteady flows is now feasible. However, generalisation to 3D flow fields is still problematic. Techniques based on integration for generating geometries and particle animation (see Section 4) are also very successful, and generalise better to 3D fields.

One of the original key problems in flow visualisation was the direct visualisation of directional structures in a 3D field, possibly varying in time. Despite some heroic attempts, this problem has not been solved, as perceiving three spatial and three data dimensions directly seems a very tough job for the human eye and brain. At the same time, the scale of numerical flow simulations, and thus the size of the resulting data sets, continues to grow rapidly. For these reasons, simplification strategies have to be conceived, such as spatial selection (slicing, regions of interest), data dimension reduction, geometry simplification, and feature extraction.

Slicing in a 3D field reduces the problem to 2D, allowing 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 data set 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 visualisation techniques (such as using 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 data sets and interactive exploration, enabling users to trade accuracy with response time.

Feature extraction (Section 5) is selection and simplification based on content: extracting important high-level information from a data set, visualising the data from a problem-oriented point of view. This leads to a large reduction of the data size, and to fully or semiautomatic generation of simple and clear images. The techniques are generally very specific for a certain type of problem (such as vortex detection),

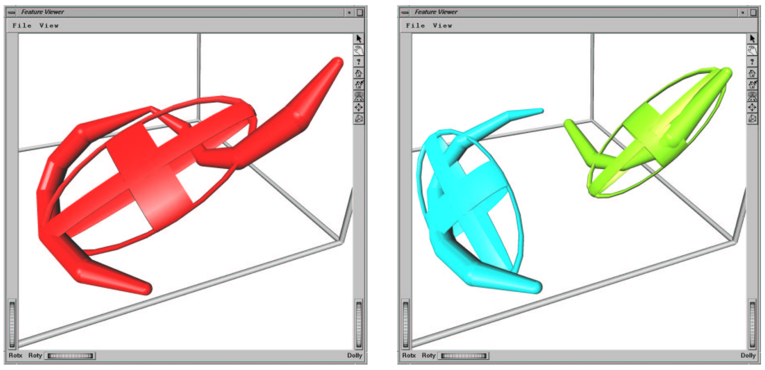


Figure 28: *A split event, before (left) and after (right). The features are visualised with both an ellipsoid and a skeleton icon* [97].

and the relation with the original raw data is indirect, and the reduction is achieved at the cost of loss of other information, which is considered irrelevant for the purpose. But the techniques generalise well to analysis of time-dependent data sets, leading to condensed episodic visual summaries.

A good possibility is combining feature extraction techniques with direct or geometric techniques. For example, selective visualisation has been used effectively with streamline generation, to place seed points in selected areas, and show important structures with only a small number of streamlines. Combining simple advection-based techniques with iconic feature visualisation can also clarify the relation between the raw data and the derived information used in feature detection. The work of visualisation and simulation experts will in the future become inseparable: the distinction between simulation and visualisation will be increasingly blurred. A good example is the tracking of phase fronts (separation between two different fluids in multifluid flows) using level set methods [124], where the feature extraction is a part of both simulation and visualisation.

How about practical application? Many techniques have been incorporated in commercial visualisation systems, including feature-based techniques [†]. The practical use of flow visualisation is most effective when visualisation experts closely cooperate with fluid dynamics experts. This is especially true in feature-based visualisation, where developing detection criteria is closely connected to the physical phenomena studied. But also other disciplines can contribute to this computational science effort: mathematicians, artists and designers, experimental scientists, image processing specialists, and also perceptual and cognitive scientists [96].

Some areas that need additional work are:

- comparative visualisation and multisource comparative data analysis
- visualisation of multivariate flow fields with scalar, vector, and tensor data
- handling and exploring huge time-dependent flow data sets
- detection and tracking of new types of features, such as surface features (shock waves, phase fronts) in time-dependent data sets
- the use of virtual environments for visual data exploration and computational steering: problems of performance and 3D interaction
- user studies for evaluation, validation, and field testing of flow visualisation techniques
- visualisation of inaccuracy and uncertainty.

Overlooking the whole landscape of flow visualisation techniques, we can say that visualisation of 2D flows has reached a high level of perfection, and for 3D a rich set of techniques is available. In the future, we will concentrate on techniques that scale well with ever increasing data set sizes, and therefore selection and simplification techniques will get more attention.

10. Acknowledgements

The authors thank all those who have contributed to this research including AVL (`www.avl.com`), the Austrian governmental research program called K plus (`www.kplus.at`), and the VRVis Research Center (`www.VRVis.at`). Furthermore, the authors thank all the permission holders of images shown in this report.

This project was partly supported by the Netherlands Organization for Scientific Research (NWO) on the NWO-EW Computational Science Project “Direct Numerical Simulation of Oil/Water Mixtures Using Front Capturing Techniques”.

[†] `http://www.ensight.com/products/flow-feature.html`

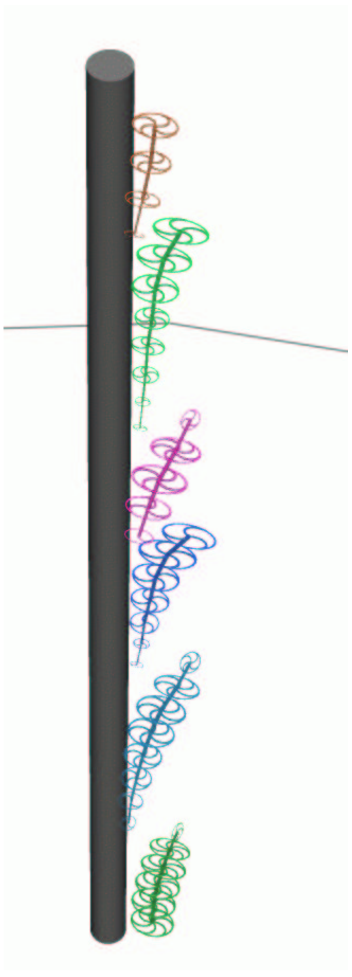
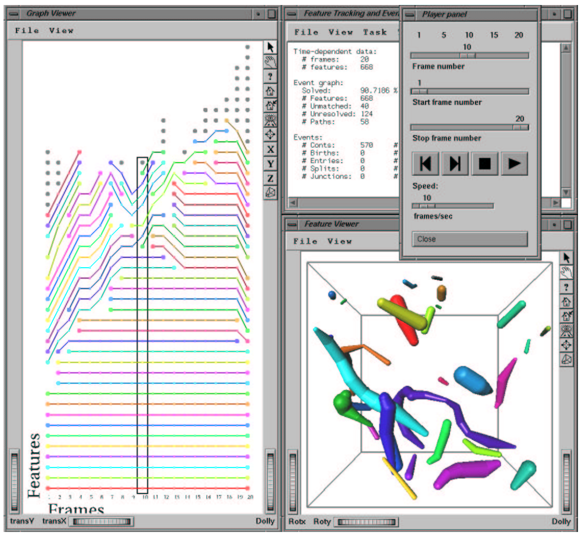


Figure 24: *3D Vortex structures behind a tapered cylinder* [97].



```
Time-dependent data:
  # frames:     20
  # features:   668

Event graph:
  Solved:       90.7186 %
  # Features:   668
  # Unmatched:  40
  # Unresolved: 124
  # Paths:      58

Events:
  # Conts:      570
  # Births:     0
  # Entries:    0
  # Splits:     0
  # Junctions:  0
```

Figure 26: *Playing through the turbulent vortex data set.*

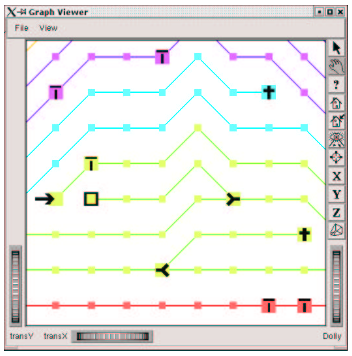


Figure 27: *Events are visualised in the graph viewer with special, characteristic icons.*

References

1. G. V. Bancroft, F. J. Merritt, T. C. Plessel, P. Kelaita, R. K. McCabe, and A. Globus. FAST: A multiprocessed environment for visualization of computational fluid dynamics. In *Proc. Visualization '90*, pages 14–27, 1990. 11

2. D. C. Banks and B. A. Singer. Vortex tubes in turbulent flows: Identification, representation, reconstruction. In *Proc. Visualization '94*, pages 132–139, 1994. 14

3. D. C. Banks and B. A. Singer. A predictor-corrector technique for visualizing unsteady flow. *IEEE TVCG*, 1(2):151–163, June 1995. 15

4. H. Battke, D. Stalling, and H. Hege. Fast line integral convolution for arbitrary surfaces in 3D. In *Visualization and Mathematics*, pages 181–195, 1997. 8

5. B. G. Becker, D. A. Lane, and N. L. Max. Unsteady flow volumes. In *Proc. Visualization '95*, 1995. 12

6. J. Becker and M. Rumpf. Visualization of time-dependent velocity fields by texture transport. In *Visualization in Sientific Computing '98*, Eurographics, pages 91–102, 1998. 7

7. S. Berger and E. Gröller. Color-table animation of fast oriented line intgral convolution for vector field visualization. In *WSCG 2000 Conference Proceedings*, pages 4–11, 2000. 7

8. E. Boring and A. Pang. Directional flow visualization of vector fields. In *Proc. Visualization '96*, pages 389–392, 1996. 6

9. M. Brill, H. Hagen, H.-C. Rodrian, W. Djatschin, and S. V. Klimenko. Streamball techniques for flow visu-

alization. In *Proc. Visualization '94*, pages 225–231, 1994. 12

10. S. Bryson and C. Levit. The virtual wind tunnel. *IEEE CGA*, 12(4):25–34, July 1992. 11

11. B. Cabral and C. Leedom. Highly parallel vector visual environments using line integral convolution. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 802–807, Feb. 1995. 3, 7

12. B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 263–272, Aug. 1993. 7

13. W. Cai and P. A. Heng. Principal stream surfaces. In *Proc. Visualization '97*, pages 75–80, 1997. 12

14. J. Clyne and J. Dennis. Interactive direct volume rendering of time-varying data. In *Data Visualization '99*, Eurographics, pages 109–120. May 1999. 6

15. 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, 1993. 6

16. M. Dindar, A. Lemnios, M. S. Shephard, K. Jansen, and D. Kenwright. Effect of tip vortex resolution on uh-60a rotor-blade hover performance calculations. In *54th AHS Annual Forum and Technology Display*. American Helicopter Society, Alexandria, Va., 1998. 16

17. D. S. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume models. In *IEEE TVCG*, volume 7(3), pages 253–264, 2001. 6

18. D. S. Ebert, R. Yagel, J. Scott, and Y. Kurzion. Volume rendering methods for computational fluid dynamics visualization. In *Proc. Visualization '94*, pages 232–239, 1994. 6

19. A. J. Fenlon and T. David. An integrated visualization and design toolkit for flexible prosthetic heart valves. In *Proc. Visualization '00*, pages 453–456, 2000. 5, 10

20. L. K. Forssell. Visualizing flow over curvilinear grid surfaces using line integral convolution. In *Proc. Visualization '94*, pages 240–247, 1994. 8

21. L. K. Forssell and S. D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE TVCG*, 1(2):133–141, June 1995. 8

22. T. Frühauf. Raycasting vector fields. In *Proc. Visualization '96*, pages 115–120, 1996. 6

23. A. L. Fuhrmann and E. Gröller. Real-time techniques for 3D flow visualization. In *Proc. Visualization '98*, pages 305–312, 1998. 12

24. T. Glau. Exploring instationary fluid flows by interactive volume movies. In *Data Visualization '99*, Eurographics, pages 277–283. May 1999. 6

25. A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology of 3d vector fields. In *Proc. Visualization '91*, pages 33–39, 1991. 15, 18

26. M. in den Haak, H. J. W. Spoelder, and F. C. A. Groen. Matching of images by using automatically selected regions of interest. In *Computing Science in the Netherlands '92*, pages 27–40, 1992. 19

27. R. Haimes. Using residence time for the extraction of recirculation regions. AIAA Paper 99-3291, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, June 1999. 19

28. R. Haimes and D. Darmofal. Visualization in computational fluid dynamics: A case study. In *Proc. Visualization '91*, pages 392–397, 1991. 17

29. H. Hege and D. Stalling. Fast LIC with piecewise polynomial filter kernels. In *Mathematical Visualization*, pages 295–314. 1998. 7

30. J. L. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer*, 22(8):27–36, Aug. 1989. 13, 14

31. J. L. Helman and L. Hesselink. Surface representations of two- and three-dimensional fluid flow topology. In *Proc. Visualization '90*, pages 6–13, 1990. 18

32. J. L. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE CGA*, 11(3):36–46, May 1991. 13, 18

33. L. Hesselink, F. H. Post, and J. van Wijk. Research issues in vector and tensor field visualization. *IEEE CGA*, 14(2):76–79, Mar. 1994. 1

34. W. Hibbard and D. Santek. Interactivity is the key. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, pages 39–43, May 1989. 3

35. J. P. M. Hultquist. Interactive numerical flow visualization using Stream Surfaces. *Computing Systems in Engineering*, 1(2-4):349–353, 1990. 12

36. J. P. M. Hultquist. Constructing Stream Surfaces in Steady 3D Vector Fields. In *Proc. Visualization '92*, pages 171–178, 1992. 12

37. J. C. R. Hunt, A. A. Wray, and P. Moin. Eddies, streams, and convergence zones in turbulent flows. In *Proceedings of the Summer Program Center for Turbulence Research*, pages 193–208, 1988. 19

38. V. Interrante and C. Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In *Proc. Visualization '97*, pages 421–424, 1997. 8, 9

39. V. Interrante and C. Grosch. Visualizing 3D flow. *IEEE CGA*, 18(4):49–53, 1998. 9

40. V. L. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 109–116, Aug. 1997. 9

41. J. Jeong and F. Hussain. On the identification of a vortex. *J. Fluid Mechanics*, 285:69–94, 1995. 15

42. M. Jiang, R. Machiraju, and D. Thompson. A novel approach to vortex core region detection. In *Data Visualization 2002. Proc. VisSym'02*, 2002. 15, 16

43. B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection. In *Proc. Visualization '00*, pages 155–162, 2000. 7

44. B. Jobard, G. Erlebacher, and M. Y. Hussaini. Tiled hardware accelerated texture advection for unsteady flow visualization. In *Proceedings of Graphicon 2000*, pages 189–196, August 2000. 7

45. B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-eulerian advection for unsteady flow visualization. In *IEEE Visualization*, 2001. 7, 8

46. B. Jobard and W. Lefer. Creating evenly–spaced streamlines of arbitrary density. In *Proc. 8th EG Workshop on Visualization in Scientific Computing*, volume 7, April 28-30 1997. 10

47. B. Jobard and W. Lefer. The motion map: Efficient computation of steady flow animations. In *Proc. Visualization '97*, pages 323–328, 1997. 7

48. B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. In *Computer Graphics Proceedings (Eurographics 2000)*, volume 19(3), 2000. 10

49. B. Jobard and W. Lefer. Multiresolution flow visualization. In *WSCG 2001, International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive Media*, February 2001. 10, 11

50. D. S. Kalivas and A. A. Sawchuk. A region matching motion estimation algorithm. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 54(2):275–288, Sept. 1991. 19

51. 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. 9

52. D. N. Kenwright. Automatic detection of open and closed separation and attachment lines. In *Proc. Visualization '98*, pages 151–158, 1998. 18

53. D. N. Kenwright and R. Haimes. Automatic vortex core detection. *IEEE CGA*, 18(4):70–74, July 1998. 16

54. D. N. Kenwright, C. Henze, and C. Levit. Feature extraction of separation and attachment lines. *IEEE TVCG*, 5(2), Apr. 1999. 19

55. D. N. Kenwright and D. A. Lane. Interactive Time-Dependent Particle Tracing Using Tetrahedral Decomposition. *IEEE TVCG*, 2(2):120–129, June 1996. 11

56. L. Khouas, C. Odet, and D. Friboulet. 2D vector field visualization using furlike texture. In *Data Visualization '99. Proc. VisSym'99*, Eurographics, pages 35–44. 1999. 7

57. R. M. Kirby, H. Marmanis, and D. H. Laidlaw. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *Proc. Visualization '99*, pages 333–340, 1999. 5

58. M. Kiu and D. C. Banks. Multi-frequency noise for LIC. In *Proc. Visualization '96*, pages 121–126, 1996. 7

59. R. V. Klassen and S. J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proc. Visualization '91*, pages 148–153, 1991. 5

60. D. A. Lane. Parallelizing a particle tracer for flow visualization. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 784–789, Feb. 1995. 3

61. D. A. Lane. *Scientific Visualization of Large-Scale Unsteady Fluid Flows*, chapter 5, pages 125–145. Scientific Visualization: Overviews, Methodologies, and Techniques. 1997. 4

62. 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, April 20-25 2002. 12

63. R. S. Laramee and R. D. Bergeron. An Isosurface Continuity Algorithm for Super Adaptive Resolution Data. In *CGI 2002, Computer Graphics International*, July 1-5 2002. (to appear). 10

64. B. van Leer. The computation of steady solutions to the euler equations: a perspective. Technical report, University of Michigan, 1986. 17

65. W. C. de Leeuw. Divide and conquer spot noise. In *Proceedings of Supercomputing'97 (CD-ROM)*, Nov. 1997. 5, 7

66. W. C. de Leeuw, H.-G. Pagendarm, F. H. Post, and B. Walter. Visual simulation of experimental oil-flow visualization by spot noise images from numerical flow simulation. In *Proc. 6th EG Workshop on Visualization in Scientific Computing*, pages 135–148, 1995. 7, 18

67. W. C. de Leeuw, F. H. Post, and R. W. Vaatstra. Visu-

alization of turbulent flow by spot noise. In *Virtual Environments and Scientific Visualization '96*, pages 286–295. Apr. 1996. 7

68. W. C. de Leeuw and R. van Liere. Comparing LIC and spot noise,. In *Proc. Visualization '98*, pages 359–366, 1998. 7

69. W. C. de Leeuw and R. van Liere. Visualization of global flow structures using multiple levels of topology. In *Data Visualization '99. Proc. VisSym'99*, pages 45–52, 1999. 13

70. W. C. de Leeuw and J. van Wijk. Enhanced spot noise for vector field visualization. In *Proc. Visualization '95*, pages 233–239, Oct. 1995. 7

71. Y. Levy, D. Degani, and A. Seginer. Graphical visualization of vortical flows by means of helicity. *AIAA Journal*, 28(8):1347–1352, Aug. 1990. 15

72. H. Löffelmann. *Visualizing Local Properties and Characteristic Structures of Dynamical Systems*. PhD thesis, Technical University of Vienna, 1998. 9

73. H. Löffelmann and E. Gröller. Enhancing the visualization of characteristic structures in dynamical systems. In *Visualization in Scientific Computing '98*, Eurographics, pages 59–68, 1998. 11

74. 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, June 1997. 7

75. 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, 1997. 12

76. H. Löffelmann, L. Mroz, E. Gröller, and W. Purgathofer. Stream arrows: Enhancing the use of streamsurfaces for the visualization of dynamical systems. *TVC*, 13:359–369, 1997. 12, 13

77. H. Löffelmann, Z. Szalavári, and M. E. Gröller. Local analysis of dynamical systems – concepts and interpretation. In *Proceedings of The Fourth International Conference in Central Europe on Computer Graphics and Visualization*, pages 170–180, 1996. 4

78. D. Lovely and R. Haimes. Shock detection from computational fluid dynamics results. In *Proceedings of the 14th AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 1999. AIAA paper 99-3285. 17

79. K.-L. Ma, J. van Rosendale, and W. Vermeer. 3d shock wave visualization on unstructured grids. In *Proc. Symposium on Volume Visualization*, pages 87–94,104, 1996. 17

80. X. Mao, Y. Hatanaka, H. Higashida, and A. Imamiya. Image-guided streamline placement on curvilinear grid surfaces. In *Proc. Visualization '98*, pages 135–142, 1998. 11

81. 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. 8

82. 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 in Boulogne-sur-Mer, France*, pages 57–70, 1997. 8

83. N. Max, B. Becker, and R. Crawfis. Flow volumes for interactive vector field visualization. In *Proc. Visualization '93*, pages 19–24, 1993. 12, 13

84. P. Moin and J. Kim. The structure of the vorticity field in turbulent channel flow. part 1. analysis of instantanious fields and statistical correlations. *J. Fluid Mechanics*, 155:441, 1985. 15

85. P. Moran, C. Henze, D. Ellsworth, S. Bryson, and D. Kenwright. The Field Encapsulation Library (FEL). 5

86. K. W. Morton and M. A. Rudgyard. Shock recovery and the cell vertex scheme for the steady euler equations. In D. L. Dowyer, M. Y. Hussaini, and R. G. Voigt, editors, *Lecture notes in physics*, pages 424–428. Springer-Verlag, 1989. 17

87. G. M. Nielson. Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE TVCG*, 5(4):360–372, Oct. – Dec. 1999. 11

88. G. M. Nielson, H. Hagen, and H. Müller. *Scientific Visualization: Overviews, Methodologies, and Techniques*. 1997. 10

89. A. Okada and D. L. Kao. Enhanced line integral convolution with flow feature detection. In *SPIE Vol. 3017 Visual Data Exploration and Analysis IV*, pages 206–217, Feb. 1997. 8

90. K. Ono, H. Matsumoto, and R. Himeno. Visualization of thermal flows in an automotive cabin with volume rendering method. In *Data Visualization 2001. Proc. VisSym'01*, pages 301–308, 2001. 6

91. H.-G. Pagendarm and B. Seitz. An algorithm for detection and visualization of discontinuities in scientific data fields applied to flow data with shock waves. In P. Palamidese, editor, *Scientific Visualization: Advanced Software Techniques*, pages 161–177. Ellis Horwood Limited, 1993. 17

92. H.-G. Pagendarm and B. Walter. Feature detection from vector quantities in a numerically simulated hypersonic

flow field in combination with experimental flow visualization. In *Proc. Visualization '94*, pages 117–123, 1994. 18

93. R. Peikert and M. Roth. The parallel vectors operator - a vector field visualization primitive. In *Proc. Visualization '99*, pages 263–270, 1999. 16

94. L. M. Portela. *On the Identification and Classification of Vortices*. PhD thesis, Stanford University, School of Mechanical Engineering, 1997. 16

95. F. H. Post and T. van Walsum. Fluid flow visualization. In *Focus on Scientific Visualization*, pages 1–40. 1993. 4

96. F. H. Post and J. J. van Wijk. Visual representation of vector fields: Recent developments and research direction. In L. R. et al., editor, *Scientific Visualization: Advances and Challenges*, chapter 23, pages 367–390. Academic Press, London, 1994. 4, 25

97. F. Reinders. *Feature-Based Visualization of Time-Dependent Data*. PhD thesis, Delft University of Technology, The Netherlands, Mar. 2001. 22, 23, 24, 25, 26

98. F. Reinders, M. E. D. Jacobson, and F. H. Post. Skeleton graph generation for feature shape description. In *Data Visualization 2000. Proc. VisSym'00*, pages 73–82, 2000. 23

99. F. Reinders, F. H. Post, and H. J. W. Spoelder. Attribute-based feature tracking. In *Data Visualization '99. Proc. VisSym'99*, pages 63–72, 1999. 20

100. F. Reinders, F. H. Post, and H. J. W. Spoelder. Visualization of time-dependent data using feature tracking and event detection. *The Visual Computer*, 17(1):55–71, Feb. 2001. 20, 21, 23

101. F. Reinders, I. A. Sadarjoen, B. Vrolijk, and F. H. Post. Vortex tracking and visualisation in flow past a tapered cylinder. *Computer Graphics Forum*, 2002. To appear. 23

102. F. Reinders, H. J. W. Spoelder, and F. H. Post. Experiments on the accuracy of feature extraction. In *Proc. 9th EG Workshop on Visualization in Scientific Computing*, pages 49–58. 1998. 14

103. C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive exploration of volume line integral convolution based on 3D-texture mapping. In *Proc. Visualization '99*, pages 233–240, 1999. 9

104. S. K. Robinson. Coherent motions in the turbulent boundary layer. *Ann. Rev. Fluid Mech.*, 23:601–639, 1991. 15

105. J. van Rosendale. Floating shock fitting via lagrangian adaptive meshes. Technical Report 94-89, Institute for Computer Applications in Science and Engineering, 1994. 17

106. M. Roth. *Automatic Extraction of Vortex Core Lines and Other Line-Type Features for Scientific Visualization*. Diss. eth no. 13673, Swiss Federal Institute of Technology, ETH Zürich, 2000. 18

107. M. Roth and R. Peikert. Flow visualization for turbomachinery design. In *Proc. Visualization '96*, pages 381–384, 1996. 14, 15

108. M. Roth and R. Peikert. A higher-order method for finding vortex core lines. In *Proc. Visualization '98*, pages 143–150, 1998. 16

109. S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proc. Visualization '00*, pages 109–116, 2000. 10

110. I. Sadarjoen and F. H. Post. Deformable surface techniques for field visualization. In *Proceedings of Eurographics '97*, volume 16(3), pages C–109–C–116, 1997. 19

111. 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, 1998. 11

112. I. A. Sadarjoen and F. H. Post. Geometric methods for vortex detection. In *Data Visualization '99. Proc. VisSym'99*, pages 53–62, 1999. 16, 23

113. I. A. Sadarjoen and F. H. Post. Detection, quantification, and tracking of vortices using streamline geometry. *Computers & Graphics*, 24(3):333–341, June 2000. 17

114. I. A. Sadarjoen and F. H. Post. Techniques and applications of deformable surfaces. In H. Hagen, G. M. Nielson, and F. H. Post, editors, *Scientific Visualization, Proceedings Dagstuhl '97*, pages 277–286. IEEECS, 2000. 22, 23

115. R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *IEEEC*, 27(7):20–27, July 1994. 20, 21

116. A. Sanna, B. Montrucchio, and R. Arinaz. Visualizing unsteady flows by adaptive streaklines. In *WSCG 2000 Conference Proceedings*, 2000. 10

117. A. Sanna, B. Montrucchio, and P. Montuschi. A survey on visualization of vector fields by texture-based methods. *Research Developments in Pattern Recognition*, 1(1), 2000. 6

118. G. Scheuermann, H. Burbach, and H. Hagen. Visualizing planar vector fields with normal component using

line integral convolution. In *Proc. Visualization '99*, pages 255–262, 1999. 8

119. G. Scheuermann, H. Kruger, M. Menzel, and A. P. Rockwood. Visualizing nonlinear vector field topology. *IEEE TVCG*, 4(2):109–116, Apr. 1998. 13
120. W. Schroeder, C. R. Volpe, and W. E. Lorensen. The stream polygon: A technique for 3D vector field visualization. In *Proc. Visualization '91*, pages 126–132, 1991. 12
121. W. J. Schroeder, K. M. Martin, and W. E. Lorensen. *The Visualization Toolkit*. 2nd edition, 1998. 5, 10
122. M. Schulz, F. Reck, W. Bartelheimer, and T. Ertl. Interactive visualization of fluid dynamics simulations in locally refined cartesian grids. In *Proc. Visualization '99*, pages 413–416, 1999. 5
123. I. K. Sethi, N. V. Patel, and J. H. Yoo. A general approach for token correspondence. *Pattern Recognition*, 27(12):1775–1786, Dec. 1994. 20
124. J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2 edition, June 1999. 25
125. H. Shen and D. L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE TVCG*, 4(2), Apr. – June 1998. 8
126. H. W. Shen, C. R. Johnson, and K. L. Ma. Visualizing vector fields using line integral convolution and dye advection. In *Proc. Symposium on Volume Visualization*, pages 63–70, 1996. 7
127. H. W. Shen and D. L. Kao. UFLIC: A line integral convolution algorithm for visualizing unsteady flows. In *Proc. Visualization '97*, pages 317–323, 1997. 8
128. 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, Nov. 1990. 12
129. D. Silver, F. Post, and I. Sadarjoen. *Flow Visualization*, volume 7 of *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 640–652. 1999. 4
130. D. Silver and X. Wang. Volume tracking. In *Proc. Visualization '96*, pages 157–164, 1996. 19
131. D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE TVCG*, 3(2), June 1997. 19
132. D. Silver and X. Wang. Visualizing evolving scalar phenomena. *Future Generation Computer Systems*, 15(1):99–108, Feb. 1999. 19
133. B. Singer and D. Banks. A predictor-corrector scheme for vortex identification. ICASE Report No. 94-11, ICASE, Mar. 1994. 15
134. D. Stalling. LIC on surfaces. In *Texture Synthesis with Line Integral Convolution*, pages 51–64, 1997. 8
135. D. Stalling and H. Hege. Fast and resolution independent line integral convolution. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 249–256, August 1995. 7
136. D. Sujudi and R. Haimes. Identification of swirling flow in 3d vector fields. AIAA Paper 95-1715, June 1995. 15
137. J. E. Swan, M. Lanzagorta, D. Maxwell, E. Kou, J. Uhlmann, W. Anderson, H. Shyu, and W. Smith. A computational steering system for studying microwave interactions with missile bodies. In *Proc. Visualization '00*, pages 441–444, 2000. 6
138. Y. Tamura, K. Fujii, and T. Ogawa. Parallel computations and visualizations of flows around high-speed trains. In *Mechatronics and supercomputing applications in the transportation industries: ISATA International Symposium on Automotive Technology and Automation (27th: 1994: Aachen, Germany)*, pages 771–778, 1994. 3
139. C. K. Tang and G. G. Medioni. Extremal feature extraction from 3D vector and noisy scalar fields. In *Proc. Visualization '98*, pages 95–102, 1998. 10
140. C. Teitzel and T. Ertl. New approaches for particle tracing on sparse grids. In *Data Visualization '99*, Eurographics, pages 73–84. May 1999. 12
141. 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, 1997. 11
142. C. Teitzel, R. Grosso, and T. Ertl. Line integral convolution on triangulated surfaces. In *Proceedings of the Fifth International Conference in Central Europe on Computer Graphics and Visualization '97*, number 8, pages 572–581, 1997. 8
143. C. Teitzel, R. Grosso, and T. Ertl. Particle tracing on sparse grids. In *Visualization in Sientific Computing '98*, Eurographics, pages 81–90, 1998. 11
144. A. Telea and J. van Wijk. Simplified representation of vector fields. In *Proc. Visualization '99*, pages 35–42, 1999. 7
145. L. A. Treinish. Multi-resolution visualization techniques for nested weather models. In *Proc. Visualization '00*, pages 513–516, 2000. 6, 10
146. G. Turk and D. Banks. Image-guided streamline placement. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 453–460, Aug. 1996. 10, 11

147. S. Ueng, C. Sikorski, and K. Ma. Out-of-Core Streamline Visualization on Large Unstructured Meshes. *IEEE TVCG*, 3(4):370–?, Oct. 1997. 3

148. S. K. Ueng, C. Sikorski, and K. L. Ma. Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids. *IEEE TVCG*, 2(2):100–110, June 1996. 12

149. V. Verma, D. Kao, and A. Pang. PLIC: Bridging the gap between streamlines and LIC. In *Proc. Visualization '99*, pages 341–348, 1999. 8, 9

150. V. Verma, D. Kao, and A. Pang. A flow-guided streamline seeding strategy. In *Proc. Visualization '00*, pages 163–170, 2000. 10

151. J. Villasenor and A. Vincent. An algorithm for space recognition and time tracking of vorticity tubes in turbulence. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 55(1):27–35, Jan. 1992. 15

152. T. van Walsum. *Selective Visualization on Curvilinear Grids*. PhD thesis, Delft University of Technology, The Netherlands, Dec. 1995. 14

153. T. van Walsum, F. H. Post, D. Silver, and F. J. Post. Feature extraction and iconic visualization. *IEEETVCG*, 2(2):111–119, June 1996. 22

154. 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, Apr. 1996. 3

155. 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 *Data Visualization 2001. Proc. VisSym'01*, pages 25–34, 2001. 10

156. R. Wegenkittl and E. Gröller. Fast oriented line integral convolution for vector field visualization via the Internet. In *Proc. Visualization '97*, pages 309–316, 1997. 7

157. R. Wegenkittl, E. Gröller, and W. Purgathofer. Animating flow fields: Rendering of oriented line integral convolution. In *Computer Animation '97 Proceedings*, pages 15–21, June 1997. 7

158. R. Wegenkittl, E. Gröller, and W. Purgathofer. Visualizing the dynamical behavior of Wonderland. *IEEE CGA*, 17(6):71–79, Nov./Dec. 1997. 10

159. C. Weigle and D. C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *Proc. Symposium on Volume Visualization*, pages 103–110, 1998. 19

160. R. Westermann. The rendering of unstructured grids revisited. In *Data Visualization 2001. Proc. VisSym'01*, pages 65–74, 2001. 5, 6

161. R. Westermann, C. Johnson, and T. Ertl. Topology-preserving smoothing of vector fields. In *IEEE TVCG*, volume 7(3), pages 222–229, 2001. 12

162. J. J. van Wijk. Spot noise-texture synthesis for data visualization. In *Computer Graphics. Proc. SIGGRAPH 91*, pages 309–318, 1991. 6

163. J. J. van Wijk. Flow visualization with surface particles. *IEEE CGA*, 13(4):18–24, July 1993. 12

164. J. J. van Wijk. Implicit Stream Surfaces. In *Proc. Visualization '93*, pages 245–252, 1993. 12

165. J. J. van Wijk. Image based flow visualization. In *Proceedings SIGGRAPH 2002*, 2002. 24

166. C. Yang, T. Mitra, and T. Chiueh. On-the-fly rendering of losslessly compressed irregular volume data. In *Proc. Visualization '00*, pages 101–108, 2000. 3

167. L. A. Yates and G. T. Chapman. Streamlines, vorticity lines, and vortices. Technical Report AIAA-91-0731, American Institute of Aeronautics and Astronautics, 1991. 15

168. N. J. Zabusky, O. N. Boratav, R. B. Pelz, M. Gao, D. Silver, and S. P. Cooper. Emergence of coherent patterns of vortex stretching during reconnection: A scattering paradigm. *Physical Review Letters*, 67(18):2469–2472, 1991. 15

169. M. Zöckler, D. Stalling, and H. Hege. Interactive visualization of 3D-vector fields using illuminated streamlines. In *Proc. Visualization '96*, pages 107–113, 1996. 11, 13

170. M. Zöckler, D. Stalling, and H. Hege. Parallel Line Integral Convolution. *Parallel Computing*, 23(7):975–989, July 1997. 3, 7