Volume 23 (2004), number 2 pp. 203–221 COMPUTER GRAPHICS *forum*

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

Robert S. Laramee,[1] Helwig Hauser,[1] Helmut Doleisch,[1] Benjamin Vrolijk,[2] Frits H. Post[2] and Daniel Weiskopf[3]

[1] VRVis Research Center, Austria
[2] Delft University of Technology, Netherlands
[3] VIS, University of Stuttgart, Germany

Abstract
Flow visualization has been a very attractive component of scientific visualization research for a long time. Usually very large multivariate datasets require processing. These datasets often consist of a large number of sample locations and several time steps. The steadily increasing performance of computers has recently become a driving factor for a reemergence in flow visualization research, especially in texture-based techniques. In this paper, dense, texture-based flow visualization techniques are discussed. This class of techniques attempts to provide a complete, dense representation of the flow field with high spatio-temporal coherency. An attempt of categorizing closely related solutions is incorporated and presented. Fundamentals are shortly addressed as well as advantages and disadvantages of the methods.

Keywords: visualization, flow visualization, vector field visualization, texture-based flow visualization.

ACM CCS: I.3 [Computer Graphics]: visualization, flow visualization, computational flow visualization

1. Introduction

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

Bringing many of those solutions in linear order (as necessary for a text like this) is neither easy nor intuitive. Several options of subdividing this broad field of literature are possible. Hesselink *et al.*, for example, addressed the problem of how to categorize techniques in their 1994 overview of research issues [24] and consider dimensionality as a means to classify the literature. In the following, several aspects are discussed on an abstract level before literature is addressed directly.

1.1. Classification

According to the different needs of the users, there are different approaches to flow visualization (cf. Figure 1):

- *Direct flow visualization:* This category of techniques uses a translation that is as direct as possible for representing flow data in the resulting visualization. The result is an overall picture of the flow. Common approaches are drawing arrows (Figure 2, left) or color coding velocity. Intuitive pictures can be provided, especially in the case of two dimensions. Solutions of this kind allow immediate investigation of the flow data.
- *Dense, texture-based flow visualization:* Similar to direct flow visualization, a texture is computed that is used to generate a dense representation of the flow (Figure 2, middle). A notion of where the flow moves is incorporated through corelated texture values along the vector field. In most cases this effect is achieved through filtering of texture values according to the local flow vector.

Submitted December 2002
Revised February 2003
Accepted March 2004

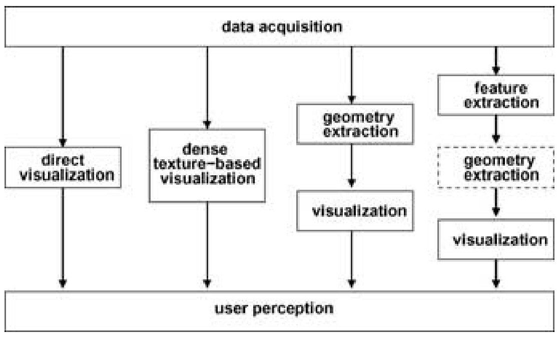


Figure 1: *Classification of flow visualization techniques—(left) direct, (middle-left) texture-based, (middle-right) based on geometric objects and (right) feature-based.*

- *Geometric flow visualization:* For a better communication of the long-term behavior induced by flow dynamics, *integration-based approaches* first integrate the flow data and use geometric objects as a basis for flow visualization. The resulting integral objects have a geometry that reflects the properties of the flow. Examples include streamlines (Figure 2, right), streaklines and pathlines. These geometric objects are based on integration as opposed to other geometric objects, like isosurfaces, that may also be useful for visualization. A description of geometric techniques is presented by Post *et al.* [55].
- *Feature-based flow visualization:* Another approach makes use of an abstraction and/or extraction step which is performed before visualization. Special features are extracted from the original dataset, such as important phenomena or topological information of the flow. Visualization is then based on these flow features (instead of the entire dataset), allowing for compact and efficient flow visualization, even of very large and/or time-dependent datasets. This can also be thought of as visualization of *derived* data. Post *et al.* [56] cover feature-based flow visualization in detail.

Figure 1 illustrates a classification of the aforementioned classes and Figure 2 shows three typical examples. Note that there are different amounts of computation associated with each category. In general, direct flow visualization techniques require less computation than the other three categories, whereas feature-based techniques require the most computation. This overview focuses on the body of research related to dense, texture-based techniques.

1.2. Spatial temporal and data dimensionality

Solutions in flow visualization differ with respect to the dimensionality of the flow data. Useful techniques for 2D flow data, like color coding or arrow plots, sometimes lack similar advantages in 3D. Here, the spatial dimensionality of the flow data is indicated as either 2D, 2.5D or 3D. In our classification the dimensionality of the results from each technique is marked with a corresponding label indicating dimensionality (see Figure 4).

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

In addition to *spatial* dimension, *temporal* dimension (dimensionality with respect to time) is of great importance. Firstly, velocity incorporates a notion of time—flows are often interpreted as differential data with respect to time (cf. Equation 1), i.e. when integrating the data, instantaneous paths such as streamlines may be obtained (cf. Equation 3). We call this *steady velocity time*. Additionally, the flow data itself can change over time resulting in time-dependent (or unsteady) flow. We refer to this as *unsteady velocity time*. The visualization must carefully distinguish between both. Performing integration in the case of unsteady data results in pathlines or streaklines as opposed to streamlines.

The distinction between steady and unsteady velocity time is important especially when animation is used in the visualization. Then, even a third notion of time, i.e. animation time, may affect the visualization. Animation time can be an arbitrary feature added to the visualization in order to create motion. Sometimes, geometric objects like streamlines are animated in order to show flow orientation, e.g. the motion of color controlled by a color table [31]. Animation is also often added to texture-based methods with the same goal in mind. Special attention is required for correct interpretation of animation time.

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

Although we do not have space to focus on experimental flow visualization, it is interesting to recognize that many computational solutions more or less mimic the visual appearance of well-accepted techniques in experimental visualization (cf. particle traces, dye injection or Schlieren techniques [77]).

1.3. Data sources

Computational flow visualization, in general, deals with data that exhibits temporal dynamics like the results from (a) *flow simulation* (e.g. the simulation of fluid flow through a turbine), (b) *flow measurements* (possibly acquired through

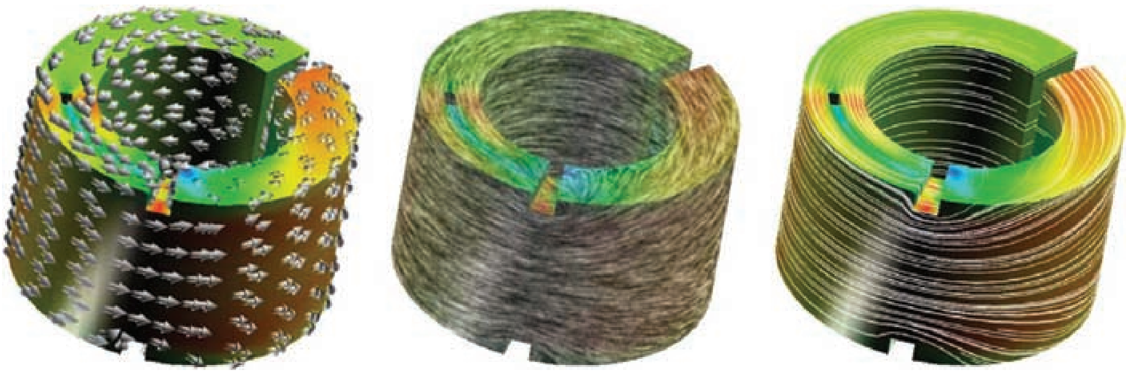


Figure 2: *An example of circular flow at the surface of a ring to help illustrate our flow visualization classification: (left) direct visualization by the use of arrow glyphs, (middle) texture-based by the use of LIC and (right) visualization based on geometric objects, here streamlines.*

laser-based technology) or (c) *analytic models* of flows (e.g. dynamical systems [1], given as set of differential equations).

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

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

2. Fundamentals

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

Flow simulations are often solutions to systems of PDEs, such as the Navier Stokes, Euler or Advection-Diffusion equations [82]. In general, discretized solution methods are used. Noteworthy are finite volume (FV) and finite element (FE) analysis, which subdivide the domain into small elements like hexahedral or tetrahedral cells. A solution is defined on the computation grid in physical space: unstructured for FE and structured curvilinear for FV solutions. In the discussion that follows, we assume that vector data are defined on the grid nodes (cell vertices).

2.1. Reconstruction of flow data

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

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

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

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

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

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

2.2. Grids

In flow simulation, the vector samples $\mathbf{v}_i$ usually are laid out across the flow domain with respect to a certain type of grid. Grid types range from simple rectilinear or Cartesian grids to curvilinear grids to complex unstructured grids (cf. Figure 3). Typically, simulation grids exhibit large variations in cell sizes. This variety of cell sizes stems from the influence

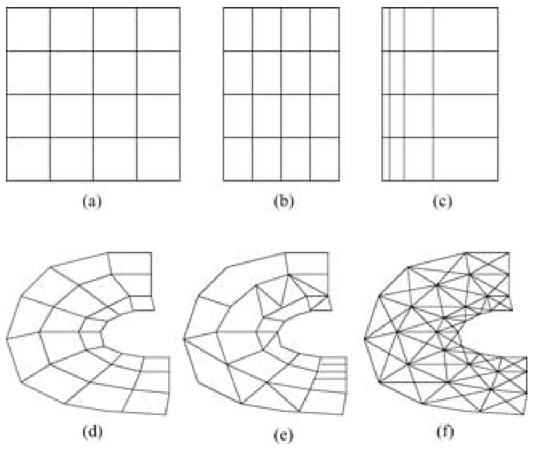


Figure 3: *Grids involved in flow simulation—(a) Cartesian, (b) regular, (c) general rectilinear, (d) structured or curvilinear, (c) unstructured and (f) unstructured triangular [37,89].*

of grid generation onto the flow simulation process. The quality of the grid model and its implementation impact the quality of the simulation results.

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

Starting with *point location*, i.e. the problem of finding the grid cell in which a given nD-point lies, usually two cases are distinguished. For general point location, special data structures can be used that subdivide the spatial domain to speed up the search. For iterative point location, often needed during integral curve computation, algorithms are used that efficiently exploit spatial coherence during the search. One kind of such algorithms starts with an initial guess for the target cell, checks for point containment and refines accordingly afterward. This process is iterated until the target cell is found. More details can be found in text books about flow visualization fundamentals [53,68].

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

After point location and flow reconstruction, visualization begins: vectors can be represented with glyphs; virtual particles can be injected and traced across the flow domain. Nevertheless, the *computation of derived data* may be necessary to do more sophisticated flow visualization. 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, or flow rotation and shear. For feature extraction, flow vorticity $\omega = \nabla \times \mathbf{v}$ can be of high interest. Further details about local flow properties can be found in previous work [45,54].

2.3. Integration

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

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

where $\mathbf{p}_0$ represents the location of the particle path at seed time 0. Note that Equations 1 and 3 are complimentary to each other. For other types of integral curves, such as streaklines see previous work [36,68].

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* are employed. The most simple approach is to use a first-order Euler method to compute an approximation $\mathbf{p}_E(t)$—one iteration of the curve integration is specified by

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

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 [57], which uses the Euler approximation $\mathbf{p}_E$ as a look-ahead to compute a better approximation $\mathbf{p}_{RK2}(t)$ of the integral curve:

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

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

3. Dense and Texture-Based Flow Visualization

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

- *Spot Noise techniques:* These methods (Section 3.1) are based on a technique introduced by Van Wijk [78]. In this category, the basic primitive on which the algorithms operate is the so-called *spot*: an ellipse or other shape that is warped and distributed in order to reflect the characteristics of a vector field.
- *LIC techniques:* The methods in this category (Section 3.2) are derived from an algorithm introduced by Cabral and Leedom [8], namely, line integral convolution (LIC). The basic primitive here is a *noise texture*: the properties of texture are convolved, or smeared, using a kernel filter in the direction of the underlying vector field.
- *Texture advection and GPU-based techniques:* The primitive in this case (Section 3.3) is a *moving texel* [50]. Individual texels/texel properties, or groups of texels are advected in the direction of the vector field. Many of the techniques in this category utilize more computation on the GPU (Graphics Processing Unit)—rather than the CPU—in order to realize performance gains.
- *Related techniques:* Most of the dense, texture-based flow visualization research falls into one of the previous categories. Related research that does not fit cleanly into one of the previous classifications is discussed in Section 3.4.

We have included a section of meta-research papers in Section 4 after the individual research techniques. These papers attempt to provide an alternative, higher-level framework that incorporates many of the techniques discussed here.

3.1. Spot noise

Spot noise, introduced by Van Wijk [78], was one of the first dense, texture-based techniques for vector field visualization. Spot noise generates a texture by distributing a set of intensity functions, or spots, over the domain. Each spot represents a particle warped over a small step in time and results in a streak in the direction of the local flow from where the particle is seeded. A spot noise texture is defined by: [78]

$$f(\mathbf{x}) = \sum a_i h(\mathbf{x} - \mathbf{x}_i, \mathbf{v}(\mathbf{x}_i)) \tag{6}$$

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

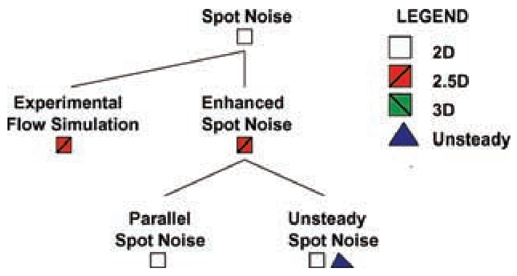


Figure 4: *The spot noise hierarchy of related research. Children in the hierarchy build upon the work of their parent.*

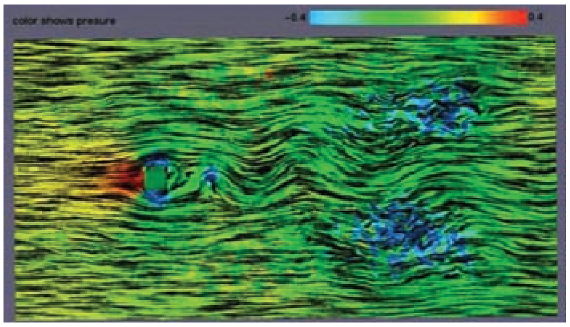


Figure 5: *A snapshot of the unsteady spot noise algorithm [16]. Image courtesy of De Leeuw and Van Liere.*

The hierarchy shown in Figure 4 illustrates the relationship amongst spot noise related methods. Follow-up research that builds upon a previous technique is shown as a child in the hierarchy. Children that share a common parent are presented in chronological order of appearance when reading from left to right. Each node in the hierarchy is labeled and the corresponding description can be matched in the text of this article. The dimensionality of the flow data used to generate the results is indicated for convenience. The time dimension label is given a different shape to distinguish it from the spatial dimensions. We believe the spot noise hierarchy (Figure 4) and the LIC hierarchy (Figure 7) will be valuable assets in helping the reader navigate the related research literature. In what follows, we visit each node in the hierarchy in depth-first-search order.

Comparative visualization: Spot noise has been used to simulate the results from the field of experimental flow visualization [14]. First, the parameters of the spot noise technique are tuned in order to simulate the smearing of oil on a surface. A postprocessing step is then added to enhance the visualization result such that it looks closer to the smearing of real oil from experimental flow visualization.

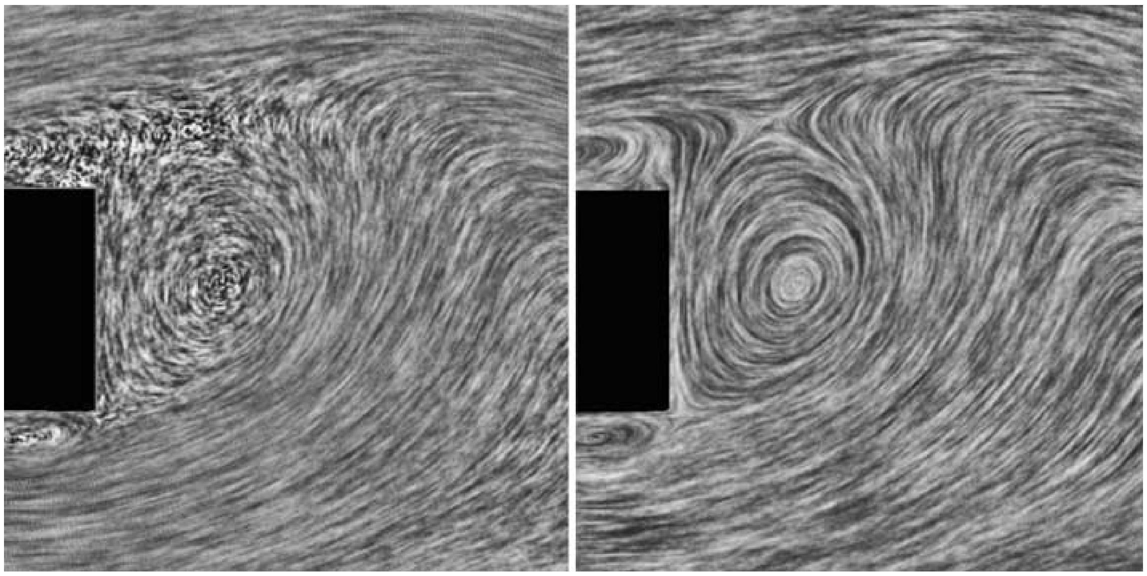


Figure 6: *Visualization of flow past a box using (left) spot noise and (right) LIC.*

Enhanced spot noise: One limitation of the original spot noise algorithm was the inability to represent high, local velocity curvature especially with high speeds. Enhanced spot noise [12] by De Leeuw and Van Wijk addresses these challenges through the use of bent spot primitives.

Parallel and unsteady spot noise: In order to accelerate the performance of enhanced spot noise towards interactive frame rates, a parallel implementation of the algorithm was introduced by De Leeuw [13]. The parallel implementation was applied to the steering of a smog prediction simulation and searching a very large data set resulting from a numerical simulation of turbulence.

The first application of spot noise to unsteady flow is presented by De Leeuw and Van Liere [16] (Figure 5). The motion of spots is modeled after particles in unsteady flow. In order to visualize unsteady flow, the distribution of spots with respect to the temporal domain is discussed. Unsteady spot noise also introduces support for zooming views of the vector field. Spot noise with zooming is also utilized by De Leeuw and Van Liere when visualizing topological features of 2D flow [10].

Spot noise related literature: A combination of both texture-based FlowVis on 2D slices and 3D arrows for 3D flow visualization is employed by Telea and Van Wijk [74]. Arrows denote the main characteristics of the 3D flow after clustering and a 2D slice with spot noise visualization serves as context. The focus of this work is on vector field clustering.

Löffelmann *et al.* [44] use anisotropic spot noise created from a grid-shaped spot to visualize streamlines and timelines concurrently on stream surfaces. Another interesting application of spot noise is its use for the depiction of discrete maps (noncontinuous flow) [43].

Spot noise has also been applied to the visualization of turbulent flow [15] and in combination with the visualization of flow topology [10,11]. We refer the reader to Post *et al.* [55,56] for more on the subject of flow topology.

Spot noise versus LIC: A visual comparison of LIC (the focus of the next section) and spot noise is shown in Figure 6. Spot noise is capable of reflecting velocity magnitude within the amount of smearing in the texture, thus freeing up hue for the visualization of another attribute such as pressure, temperature, etc. On the other hand, LIC is more suited for the visualization of critical points which is a key element in conveying the flow topology. The vector magnitudes are normalized thus retaining lower spatial frequency texture in areas of low velocity magnitude. De Leeuw and Van Liere also compare spot noise to LIC [17]. They report that LIC is better at showing direction than spot noise, but it does not encode velocity magnitude. By flow direction, we refer to the path along which a massless particle follows when injected into the flow.

3.2. Line integral convolution

Line integral convolution (LIC), first introduced by Cabral and Leedom [8], has spawned a large collection of research as indicated in Figure 7. The original LIC method takes as input a vector field on a 2D, Cartesian grid and a white noise texture of the same size. Texels are *convolved* (or correlated) along the path of streamlines using a *filter kernel* in order to create a dense visualization of the flow field. More specifically, given a streamline σ, LIC consists of calculating the intensity I for

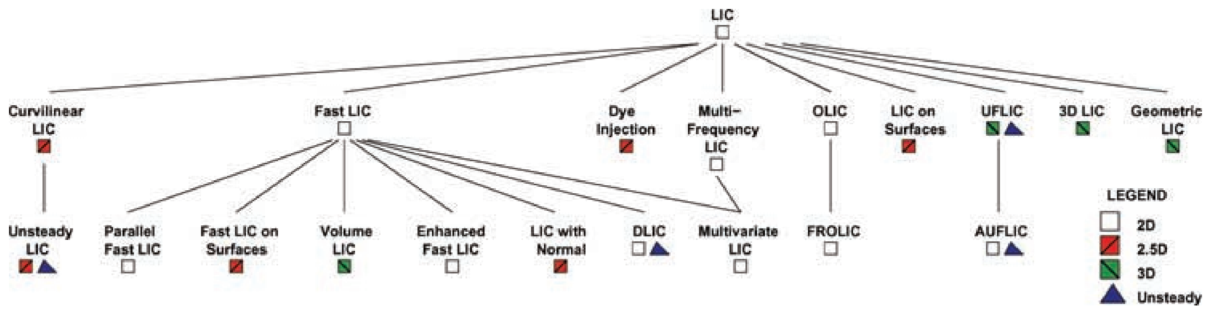


Figure 7: *The LIC hierarchy of related research. Node labels correspond to paragraphs in the text, which then lead to specific entries in the bibliography.*

a pixel located at $\mathbf{x}_0 = \boldsymbol{\sigma}(s_0)$ by: [70]

$$I(\mathbf{x}_0) = \int_{s_0-L/2}^{s_0+L/2} k(s - s_0)T(\boldsymbol{\sigma}(s))\,ds \qquad (7)$$

where T stands for an input noise texture, k denotes the filter kernel, s is an arc length used to parameterize the streamline curve and L represents the filter kernel length. See Figure 2 (middle) for a result. LIC was one of the first dense, texture-based algorithms able to accurately reflect velocity fields with high local curvature.

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

Curvilinear grids and unsteady LIC: Forssell [20] was early to extend LIC to surfaces represented by curvilinear grids. The original LIC method portrays a vector field with uniform velocity magnitude. Forssell introduces a technique for displaying vector magnitude. She also describes one approach to animate the resulting LIC textures. Forssell and Cohen extend this work to visualize unsteady flow [21]. Their approach modifies the convolution such that the filter kernel operates on streaklines rather than streamlines. In other words, they modify the LIC algorithm to trace a path that incorporates multiple time steps.

Fast LIC: Many algorithms are built on fast LIC introduced by Stalling and Hege [70]. Fast LIC is approximately one order of magnitude faster than the original. The speedup is achieved through two key observations: (1) fast LIC minimizes the computation of redundant streamlines present in the original method and (2) fast LIC exploits similar convolution integrals along a single streamline and thus re-uses parts of the convolution computation from neighboring streamline texels. They also introduce support for filtered images at arbitrary resolution.

Parallel fast LIC: Amongst the first parallel implementations of fast LIC is that of Zöckler *et al.* [90]. The proposed algorithm computes animation sequences on a massively parallel distributed memory computer. Parallelization is performed in image space rather than in time in order to take advantage of the strong temporal coherence between frames. Luckily, as we shall see later, flow visualization research in this area has evolved far enough such that expensive parallel processing hardware is not always necessary to achieve interactive visualization [28,29,38,79]. However, for 3D and unsteady flow there is still need for parallelization. For the sake of completeness, we also mention the work of Cabral and Leedom on parallelization of LIC [7] although this is a parallel processing version of the original LIC algorithm, not fast LIC.

Fast LIC on surfaces: Battke *et al.* [2] extend fast LIC to surfaces represented by arbitrary grids in 3D. The approach by Forssell and Cohen [21] was limited to surfaces represented by curvilinear grids. The method works by tessellating a given surface representation with triangles. The triangles are packed (or tiled) into texture memory and a local LIC texture is computed for each triangle. The results presented here are limited to relatively small simple surface representations composed of equilateral triangles (1,600–4,000 triangles).

Volume LIC: Interrante and Grosch [25,26] visualize true 3D flow using the fast LIC algorithm as a starting point. Clearly, there are perceptual challenges related to 3D flow visualization such as occlusion, depth perception and visual complexity. Volume LIC introduces the use of *halos* in order to enhance depth perception such that the user has a better chance at perceiving the 3D space covered in the visualization (Figure 8). Areas of higher velocity magnitude are mapped to higher texture opacity. It is interesting to note that with the introduction of halos, we are then able to identify distinct entities in the 3D field, a property generally not present in other LIC techniques. Thus, the 3D LIC takes a step in

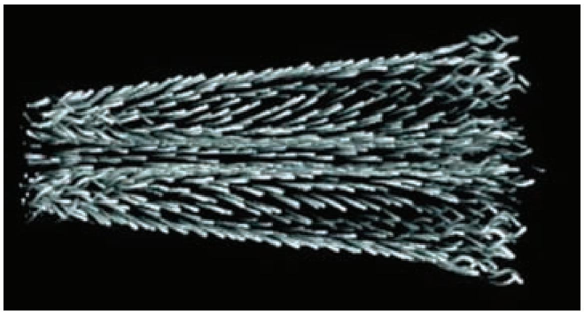


Figure 8: *A result from the Volume LIC method [25,26]. Image courtesy of Interrante and Grosch.*

the direction of being a geometric flow visualization technique where discrete integral objects such as streamlines can be distinguished. Without introducing some notion of sparseness into the visualization, the results would not be very useful. However, with the introduction of sparseness, a trade-off is made between flow field coverage and reducing occlusion.

Enhanced fast LIC and LIC with normal: Two useful extensions to the fast LIC algorithm are introduced by Hege and Stalling [22] and Scheuermann *et al.* [64]. Hege and Stalling [22] experiment with higher order filter kernels in order to enhance the quality of the resulting LIC textures.

In the case of slices, vector components orthogonal to the slice are removed when using texture-based and geometric methods for visualization results. Scheuermann *et al.* [64] address this missing orthogonal vector field component by extending fast LIC to incorporate a normal component into the visualization.

DLIC: Sundquist [71] presents an extension to fast LIC, DLIC (Dynamic LIC), in order to visualize time-dependent electromagnetic fields. According to Sundquist, the motion of the field is not necessarily along the direction of the field itself in the case of electromagnetic fields. The algorithm proposed here handles the case of when the vector field and the direction of the motion of the field lines are independent. Conceptually, there are two vector fields used in this approach: (1) the electromagnetic field itself and (2) the vector field that describes the evolution of streamlines as a function of time.

Multivariate LIC: Urness *et al.* [76] present an extension to fast LIC that incorporates a new coloring scheme that can be used to incorporate multiple 2D scalar and vector attributes. *Color weaving* assigns a specific attribute represented by a color to an individual streamline thread in the visualization. The streamline patterns may interweave and thus so may the color patterns. Using multiple colors allows visualization of more than one variate in the result. Their second contribution is called *texture stitching*: an extension to the idea presented by Kiu and Banks [34], namely multifrequency LIC. However, in the case of Urness *et al.* [76] the multifrequency noise textures are used to highlight regions of interest as opposed to velocity magnitude as by Kiu and Banks [34].

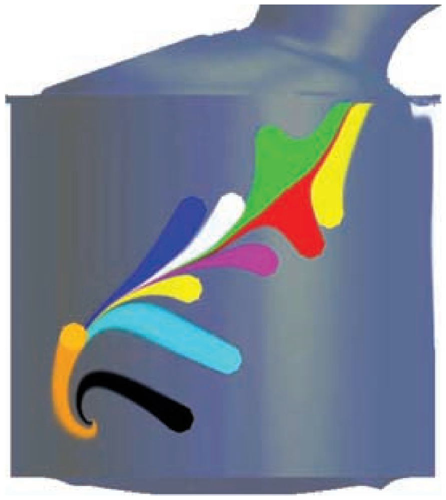


Figure 9: *Dye injection is used to highlight areas of the flow in combination on the boundary surface of an intake port and combustion chamber.*

Dye injection: Shen *et al.* address the problem of directional cues in LIC by incorporating animation and introducing dye advection into the computation [66]. The simulation of dye may be used to highlight features of the flow. In addition, they incorporate volume rendering methods that map a LIC texture onto a 3D surface. Thus the user is able to visualize the dye throughout the volume. We point out that the modeling of dye transport is not always physically correct since dye is dispersed not only by advection, but also by diffusion. Note that dye advection techniques can be classified differently. Dye injection can result in discrete geometric objects used to visualize the flow, and thus, could be classified as a group of geometric visualization techniques. Dye injection is also implemented by some of the texture advection and GPU-based techniques described in Section 3.3.

Again, in Shen *et al.* we see the notion of a sparser visualization in order to see into the 3D flow. The resulting 3D visualization approaches that of a geometric technique such as the use of streamsurfaces. And just as with the other geometric techniques, the notion of where to place or inject the dye into the flow becomes important. Figure 9 illustrates the use of dye injection.

Multifrequency LIC: Kiu and Banks propose to use a multifrequency noise for LIC [34]. The spatial frequency of the noise is a function of the magnitude of the local velocity. Long, fat streaks indicate regions of the flow with higher velocity magnitude.

One problem with many curvilinear grid LIC algorithms is that the resulting LIC textures may be distorted after being mapped onto the geometric surfaces, since a curvilinear grid usually consists of cells of different sizes. Mao *et al.* propose a solution to the problem by using multigranularity noise as the input image for LIC [46].

OLIC and FROLIC: Wegenkittl *et al.* address the problem of direction of flow in still images with their OLIC (Oriented LIC) approach [84]. By orientation, they mean the upstream and downstream directions of the flow, not visible in the original LIC implementation. Conceptually, the OLIC algorithm makes use of a sparse texture consisting of many separated spots that are smeared in the direction of the local vector field through integration. A fast version of OLIC, called FROLIC (Fast Rendering of OLIC), is achieved by Wegenkittl and Gröller [83] via a trade-off of accuracy for time. FROLIC approximates the simulated droplet trace resulting from OLIC with a sequence of disks of varying intensity, with disk intensity increasing towards the downstream direction.

Animated FROLIC [4] achieves animation of the result via a color-table and is based on the observation that only the colors of the FROLIC disks need to be changed. Each pixel is assigned a color-table index that points to a specific entry in the color-table. Color-table animation then changes the entries of the color-table itself rather than the pixels of the corresponding image.

LIC on surfaces: Mao *et al.* [47] extend the original LIC method by applying it to surfaces represented by arbitrary grids in 3D. Former LIC methods targeted at surfaces were restricted to structured grids [20,21,66]. Also, mapping a computed 2D LIC texture to a curvilinear grid may introduce distortions in the texture. Mao *el al.* propose solutions to overcome these limitations. The principle behind their algorithm relies on solid texturing [52]. The convolution of a 3D white noise image, with filter kernels defined along the local streamlines, is performed only at visible ray-surface intersections.

This idea has an advantage over that of Battke *et al.* [2] in that it avoids what can be a timely and complex assembly of triangles into texture space. However, ray-tracing is also costly. The method here is view-point dependent and required relatively lengthy processing time for an unstructured mesh composed of 10,000 triangles.

A significant body of research is dedicated to the extension of LIC onto boundary surfaces. Teitzel *et al.* [73] present an approach that works on both 2D unstructured grids and directly on triangulated grids in 3D space. This topic itself is the subject of a survey by Stalling [69].

UFLIC: Shen and Kao [67] extend the original LIC algorithm to handle unsteady flows. Their extension, called UFLIC (Unsteady Flow LIC), handles the case of unsteady flow fields by introducing a new convolution filter that better models the nature of unsteady flow. The convolution is done along pathlines (as opposed to streamlines). They improve upon the shortcomings of the previous unsteady LIC attempt presented by Forssell and Cohen [21]. According to Shen and Kao, Forssell and Cohen's approach has multiple limitations including: (1) lack of clarity with respect to spatial coherence, (2) deriving current flow values from future flow values, (3) unclear exposition with respect to temporal coherence and (4) lack of accurate time stepping. All of these problems are addressed by UFLIC. Shen and Kao also apply UFLIC to the visualization of time-dependent flow to parameterized surfaces. UFLIC is also extended using a parallel implementation by Shen and Kao [65].

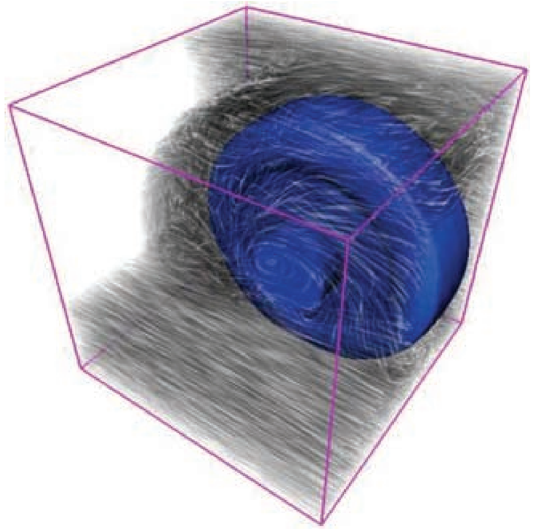


Figure 10: *An LIC visualization showing a simulation of flow around a wheel [59]. The appropriate choice of transfer function results in a sparser noise texture. Image courtesy of Rezk-Salama et al. [59].*

AUFLIC: AUFLIC (Accelerated UFLIC) is an extension to UFLIC that enhances performance times [41]. The principle behind AUFLIC is to save, reuse, and update pathlines in a vector field seeding strategy. AUFLIC requires approximately one half of the time required by UFLIC and generates similar results.

3D LIC: Rezk-Salama *et al.* [59] propose rendering methods to effectively display the results of 3D LIC computations. They utilize texture-based volume rendering in an effort to provide exploration of 3D LIC textures at interactive frame rates. Like Interrante [26], they address the perceptual problems posed by dense, 3D visualization. They approach these challenges through the use of transfer functions and clipping planes, as in Figure 10. Transfer functions allow the user to see through portions of the LIC textures deemed uninteresting by the user. In addition to conventional clipping planes,

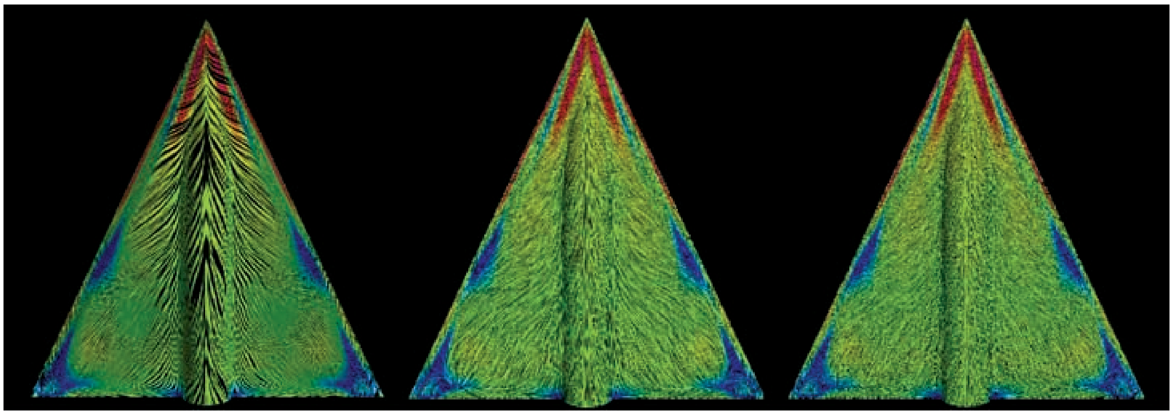


Figure 11: *A comparison of 3 LIC techniques: (left) UFLIC [65], (middle) ELIC [51], and (right) PLIC [81]. Image courtesy of Verma et al. [81].*

Rezk-Salama *et al.* also use clipping with arbitrary closed-surface geometries.

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

Geometric LIC: We make a distinction between geometric flow visualization and dense, texture-based flow visualization. However, these two topics are closely coupled. Conceptually, the path from using geometric objects to texture-based visualization is obtained via a dense seeding strategy. That is, densely seeded geometric objects result in an image similar to that obtained by dense, texture-based techniques [30]. Likewise, the path from dense, texture-based visualization to visualization using geometric objects is obtained using something such as a sparse texture for texture advection [84].

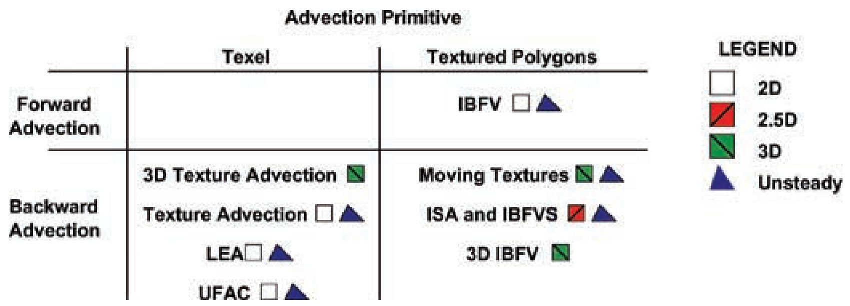
Here we have grouped together techniques that synthesize LIC results by mapping a precomputed LIC texture onto geometric primitives such as streamlines. By using geometric primitives, researchers hope to speed up performance times of the LIC results. The drawback of these methods is that they require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

Motion map: Jobard and Lefer use a *motion map* [31] in order to animate 2D steady flows. First, the domain is covered completely with streamlines. Next, a color is mapped to the streamlines and a color-table animation technique is used to animate the flow. It offers the advantage of saving memory and computation time since only one image of the flow has to be computed and stored in the motion map data structure. This technique is not applicable to unsteady flow however. It relies on a one-time cost of computing a set of streamlines.

PLIC: Verma *et al.* present a method for visual comparison of streamlines and LIC called PLIC [81] (Pseudo-LIC). They attempt to identify the relevant parameters to generate LIC-like images from a dense set of streamlines and for generating streamline-like images through the use of different filters used for convolution. By experimenting with different input textures for LIC, both streamline-like images and LIC-like results can be obtained. ELIC (enhanced LIC), placed here because of its visual comparison with PLIC, builds on the original LIC algorithm in four ways: (1) by incorporating an algorithm to improve the delineation of streamlines, (2) increasing the image contrast, (3) removing texture distortion introduced by applying LIC to curvilinear grids and (4) using color to highlight flow separation and reattachment boundaries. A visual comparison between UFLIC [65], ELIC [51], and PLIC is shown in Figure 11.

Hierarchical LIC: Bordoloi and Shen [5] introduce a hierarchical approach to LIC based on a quadtree data structure used to support level of detail (LOD). The idea is to replace portions of the vector field of lower complexity with rectangular LIC texture-mapped patches. The LIC texture is taken from a previously calculated LIC image of a straight vector field. Here, complexity is a direct function of the amount of curl in the local vector field.

Decoupled LIC: Li *et al.* [40] present a technique for the visualization of 3D flow based on texture mapped primitives, namely streamlines. They decouple the visualization into a



	Advection Primitive	
	Texel	Textured Polygons
Forward Advection		IBFV ⬜ ▲
Backward Advection	3D Texture Advection 🟩 Texture Advection ⬜ ▲ LEA⬜ ▲ UFAC ⬜ ▲	Moving Textures 🟩 ▲ ISA and IBFVS 🟥 ▲ 3D IBFV 🟩

LEGEND	
⬜	2D
🟥	2.5D
🟩	3D
▲	Unsteady

Figure 12: *The classification of texture advection and GPU-based techniques. The columns indicate the primitive used during advection while the rows indicate the advection scheme.*

preprocessing type stage that computes the streamlines and a stage which maps various textures to the streamlines computed in the first stage. The result is volume rendered at interactive frame rates. To address the perceptual challenges posed by 3D visualization, depth cues, lighting effects, silhouettes, shading and interactive volume culling are described.

3.3. Texture advection and GPU-based techniques

In this section we describe research based on moving texels or moving groups of texels, i.e. texture-mapped polygons whose motion is directed by the vector field. Figure 12 shows an overview of the different techniques and classifies them according to two properties: (1) the advection scheme used and (2) the primitive used during advection. Some of the literature focuses mainly on the integration scheme used to advect textures or texels. By the term texel means texture element. Some methods focus on the mapping to advected primitives and some focus on both. Figure 12 also shows the dimensionality of the flow data. In our discussion, we visit the methods in clockwise order starting at 12 o'clock. Within each sub-block the methods are listed in chronological order. This is because the mapping of texel properties between two time steps in the visualization is not 1-to-1 in this case. For a more detailed discussion see Jobard *et al.* [28,29].

One characteristic common to many of the texture advection techniques in this section [28,29,38,48] is the use of *backward coordinate integration* (or backward advection). None of the methods described here use forward advection (i.e. forward integration) and individual texels as a primitive. This is because the combination of forward integration and texel primitives leaves holes in the visual domain after the forward integration computation [29]. Given a position, $\mathbf{x}_0(i, j) = (i, j)$ of each particle in a 2D flow, backward integration over a time interval h determines its position at a previous time step [28]:

$$\mathbf{x}_{-h}(i, j) = \mathbf{x}_0(i, j) + \int_{\tau=0}^{h} \mathbf{v}_{-\tau}(\mathbf{x}_{-\tau}(i, j))\, d\tau \qquad (8)$$

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

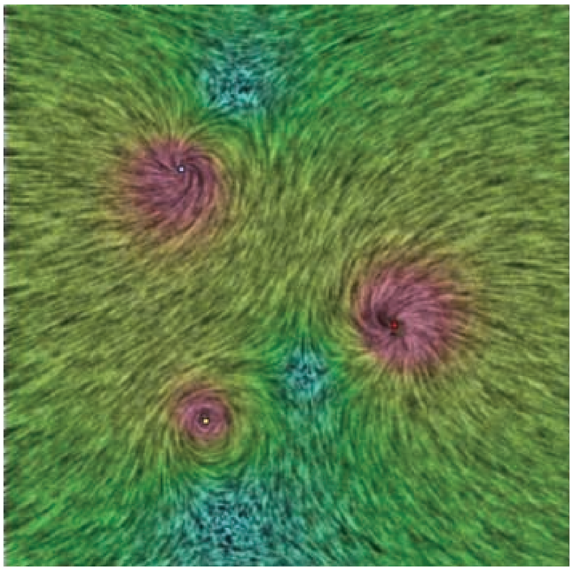


Figure 13: *A screen shot from the image based flow visualization algorithm. Image courtesy of Van Wijk [79].*

IBFV: Image based flow visualization (IBFV) by Van Wijk [79] is one of the fastest algorithms for dense, 2D, unsteady vector field representations (Figure 13). It is based on the advection and decay of textures in image space. Each frame of the visualization is defined as a blend between the previous image, warped according to the flow direction, and a number of background images composed of filtered white noise textures. One reason it is faster than many texture-based flow visualization methods is because it reduces the number of

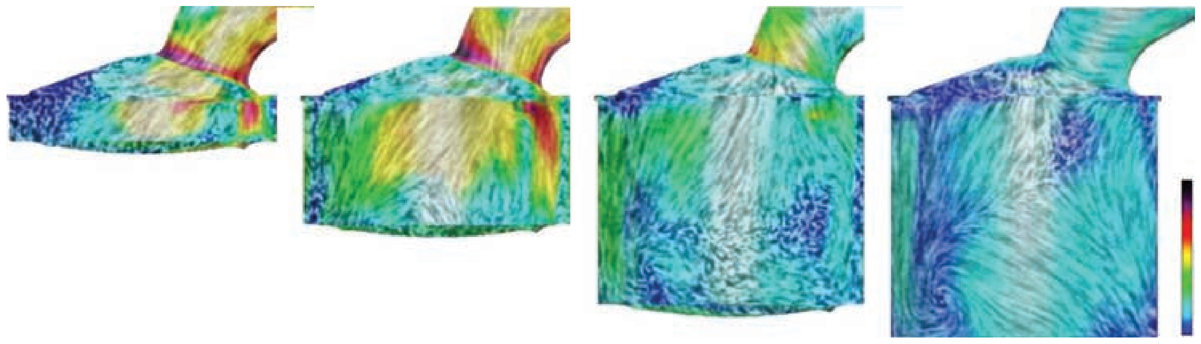


Figure 14: *Snapshots from the visualization of a time-dependent surface mesh composed of 79 K polygons with dynamic geometry and topology [38].*

integration computations that need to be performed via advecting small quadrilaterals rather than individual pixels.

Moving textures: Max and Becker were early to introduce the idea of moving textures in order to visualize vector fields [48]. One of the primary goals of this work was to use textures in motion to produce near-real-time animation of flow. Texture-mapped triangles are advected, or distorted, in the direction of the flow. Also, applying this technique to 3D flows with no modification provides results that are difficult to perceive, at least in the case of a still image.

ISA and IBFVS: IBFV has been extended to the visualization of flow on surfaces [38,80]. Van Wijk presents an extension called IBFVS, IBFV for Surfaces. Laramee *et al.* [38] present a similar dense, texture-based visualization technique on surfaces for unsteady flow called ISA (Image Space Advection). Both methods produce animated textures on arbitrary 3D triangle meshes in the same manner as the original IBFV method. Textures are generated, advected and blended in image space. The methods generate dense representations of time-dependent vector fields with high spatio-temporal correlation. While the 3D vector fields are associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space. Both spot noise and LIC-like results can be attained. In both techniques, [38,40] fast frame rates are achieved in part by exploiting the GPU.

Van Wijk's method is applied to potential field visualization and surface visualization. Laramee *et al.*'s algorithm is applied to unsteady flow on boundary surfaces of large, complex meshes from computational fluid dynamics, dynamic meshes with time-dependent geometry and topology. It has also been applied to medical simulation data as well as isosurfaces [39]. Figure 14 shows the results applied to a time-dependent geometry and topology.

3D IBFV: IBFV has also been applied to the visualization of 3D flow [75]. The problem of how to see inside the flow volume is addressed by varying both the noise sparsity, reminiscent of Interrante and Grosch [26], and through varying the opacity of the rendered volume similar to Rezk-Salama *et al.* [59]. In order to achieve sparseness, Telea and Van Wijk inject empty holes of noise into the 3D field, in addition to the noise described by the original IBFV. One important component of their method is to define a threshold value which eliminates all close-to-transparent texel values. One disadvantage of the method is that the range of velocity values it can display is limited: A texel property cannot be advected by more than one slice along the z axis of the volume in one animation frame. This problem is addressed by Weiskopf and Ertl [87].

3D Texture Advection: Kao *et al.* discuss the use of 3D and 4D texture advection for the visualization of 3D fluid flows [32]. The results show sparse texture noise in order to visualize inside the 3D vector field. Formidable challenges are introduced by the memory requirements involved in using 3D and 4D textures. The proposed method does not work well for the case of flows containing critical points for incoming flows from the grid boundary.

GPU-based LIC: Heidrich *et al.* [23] exploit pixel textures to accelerate LIC computation. Pixel textures are an OpenGL extension by SGI that provides the functionality of dependent textures in combination with multipass rendering. Heidrich *et al.*'s implementation supports 2D, steady vector fields only, and achieves sub-second computation times for LIC image generation. While this method could be categorized as a GPU-accelerated LIC technique, we position it here due to its comparability with the following texture advection techniques [27,88] that use the same proposed OpenGL extension, handle unsteady flow and thus can be considered an extension of this technique.

LEA: Jobard *et al.* [27] introduce a GPU-assisted texture advection technique for the dense visualization of 2D, unsteady flow. While the method of Max and Becker [49]

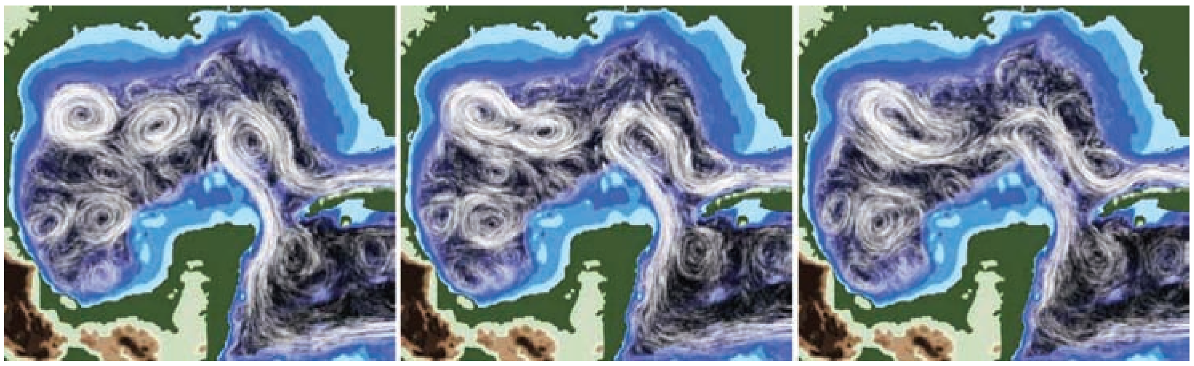


Figure 15: *Three images taken from an animation of an unsteady vector field created with the Lagrangian-Eulerian advection algorithm. [28,29] Image courtesy of Jobard* et al.

advects textures based on coarse triangular meshes, Jobard *et al.* advect textures on a per-pixel basis by means of pixel textures, which are used in a similar way as by Heidrich *et al.* [23]. The gray-scale texture from the previous time step is dragged along the flow field by modifying the texture coordinates for the dependent texture lookup according to the flow data. Nearest-neighbor sampling is combined with an update of fractional texture coordinates to represent subtexel motion and, at the same time, maintain a high contrast. An iterative injection of additional noise is used to compensate for a possible loss of contrast over time. Jobard *et al.* also discuss the treatment of inflow at boundaries, image enhancement by color masking and the use of dye advection. Because of the limited functionality of the graphics hardware that supports pixel textures, the implementation requires many rendering passes and advects a texture of size 256^2 at approximately two frames per second. Moreover, the maximum resolution of textures is restricted to 256^2.

Jobard *et al.* extend this method to the more flexible Lagrangian-Eulerian Advection (LEA) scheme [28] for the visualization of unsteady, 2D flow. Here, they rely on a CPU implementation that leads to better advection quality, higher speed, and no limitations of the maximum flow size. Particle paths are integrated as a function of time, referred to as the Lagrangian step, while the color distribution of the image pixels is stored in a texture and updated in place (Eulerian step). The temporal coherence of the advected noise textures is transformed into spatial coherence by blending textures from subsequent time steps, i.e. each still frame depicts the instantaneous structure of the flow, whereas an animated sequence of frames still reveals the motion of the advected texture. Jobard *et al.* demonstrate that the combination of noise and dye advection is useful for an effective visualization and exploration of unsteady flow. Some results from the technique are shown in Figure 15. This work is extended by Jobard *et al.* [29] in order to improve the quality of dye advection.

Weiskopf *et al.* [86] present a GPU-accelerated version of the LEA algorithm using per-fragment operations. The GPU-based texture advection by Weiskopf *et al.* [88] supports bilinear dependent texture lookups without taking into account the update of fractional coordinates. Therefore, this approach is mainly suitable for dye advection at high frame rates. Weiskopf *et al.* also demonstrate how GPU-accelerated visualization of unsteady, 3D flows can be implemented with pixel textures.

UFAC: Weiskopf *et al.* [85] introduce a generic texture-based framework for visualizing 2D, time-dependent vector fields. They propose unsteady flow advection convolution (UFAC) as an application of the framework for visualizing unsteady fluid flow. Also, their approach can reproduce other techniques such as LEA [29], IBFV [79], UFLIC [65], and DLIC [71]. Weiskopf *et al.* describe a GPU-accelerated implementation that, among other things, allows the user to trade-off quality for speed.

3.4. Related dense, texture-based methods

The literature described here is not, in general, as strongly interrelated as the literature in the spot noise, LIC, texture advection and GPU-based categories. For this reason we sought an alternative schema in order to relate the different techniques. Figure 16 shows the related methods and classifies them based on the density of their results. In this case each technique is given a subjective rating on a sparse-to-dense scale. Sparse results look more like the results from flow visualization using geometric objects whereas dense techniques produce results resembling spot noise or LIC. These methods do not fit cleanly into one of the previous categories; nonetheless, they are important to the dedicated topic and are briefly outlined here. Reading from top to bottom in Figure 16, we visit the techniques in chronological order.

Texture Splats: As an extension of the technique of splatting from volume rendering, Crawfis and Max [9] introduce the notion of *texture splats* for flow visualization. Being a volume rendering technique, it is targeted at the depiction of 3D vector fields. As with Rezk-Salama *et al.* [59], it is a selective

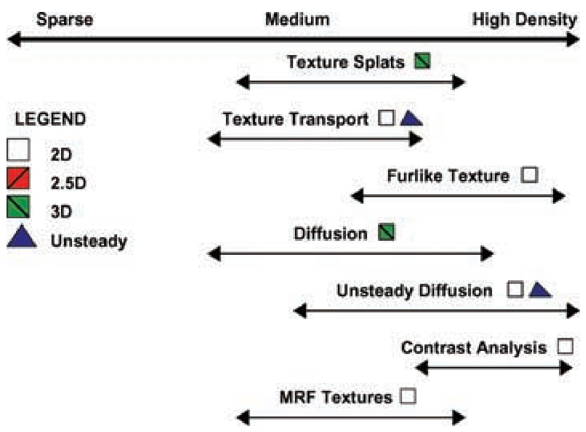


Figure 16: *Related dense, texture-based flow visualization methods. Each method is compared with respect to the density of the resulting visualization.*

transfer function that ultimately decides which subsets of the 3D data are shown and which are not. The transfer functions are used to emphasize or suppress spatial regions as opposed to ranges of data values.

Texture Transport: The texture transport method of Becker and Rumpf [3] introduces a mathematical framework based on the solution of a time-dependent transport equation. Lagrangian coordinates are computed from the transport equation and visualized using a texture mapping. The results in this case resemble those from the geometric class of solutions. Individual lines in the texture can be distinguished. The major drawback of this approach is the computation time required.

Furlike Texture: Khouas *et al.* synthesize LIC-like images in 2D with furlike textures [33]. Their technique is able to locally control attributes of the output texture such as orientation, length, density and color via a model based on filaments resembling fur.

Diffusion and Unsteady Diffusion: Preußer and Rumpf [58] as well as Diewald *et al.* [18] borrow a well known technique from image analysis for visualization of fluid flow. The non-linear, anisotropic diffusion equations from image analysis are adopted and applied to vector fields. A noisy texture covering the domain is strongly smoothed along integral lines while still retaining and enhancing edges in directions orthogonal to the flow, i.e. streamline-aligned diffusion. Successively coarse patterns representing the vector field can also be generated. It is applied to 2D, 2.5D, and 3D vector fields [18,58]. In the case of 3D, the resulting enhanced edges are discretized and resemble streamlines or streamribbons. In this case, occlusion becomes an important issue because the 3D results appear somewhat cluttered.

Bürkle *et al.* extend this technique to the case of time-dependent flow [6]. Instead of streamline-like patterns, streakline patterns are generated. A blending strategy, comparable to noise or dye injection, is introduced in order to provide the new time-dependent texture necessary for the case of long-term flow evolution. They propose a solution based on the blending of different results from the transport diffusion evolution started at successively incremented times. Again, the disadvantage of this approach is the required computational time. Also, no attempt is made to apply this method to time-dependent 3D flow, a formidable challenge.

Contrast Analysis: Sanna *et al.* [63] focus on the issue of encoding another scalar value into the texture used to visualize the flow, in addition to flow direction, orientation and local magnitude of the field. It is an extension of a previous technique called TOSL—Thick Oriented Streamline Algorithm [62]. Areas of higher scalar values are characterized by higher contrast levels in the texture and streamline tones are generated in order to highlight these areas. The goal is to allow an additional variable into the visualization beyond previous techniques.

MRF: Taponecco and Alexa apply Markov Random Field (MRF) texture synthesis methods to vector fields [72]. The results resemble a mixture of traditional texture-based methods and geometric methods. In the resulting texture, distinct streamline patterns can be seen. One drawback to this method is performance. MRF texture synthesis methods may require hours of computation time. How it may be applied to unsteady flow is an open question.

4. Comparisons and Discussion

In this section we briefly introduce literature that compares and discusses dense, texture-based techniques at a meta-level. Sanna *et al.* also provide a summary of this area of research, with a different classification [61]. The methods are classified according to the dimensionality outlined here in Section 1.2.

Flow Textures: Erlebacher *et al.* [19] present a class of flow visualization algorithms called *flow textures* within a common conceptual framework. Flow textures are textures that encode dense, 2D, time-dependent representations of flow. The framework allows important ingredients of flow texture algorithms to be understood with respect to spatial and temporal correlation. A subset of the more recent visualization techniques is described.

User Studies: Laidlaw *et al.* [35] present one of the few findings related to human-computer interaction (HCI). They attempt to assess some different visualization techniques from the viewpoint of the user in terms of searching for and classifying critical points in the flow and predicting where a particle may end after advection. Error was highest for the LIC technique in conjunction with classifying critical points and the prediction of particle advection. This is probably due to the fact that LIC images do not distinguish between upstream and downstream flow. User error was higher than expected for all methods. Hedgehog techniques and LIC were also associated with high error for locating critical points. The authors postulate that this was because in many cases critical points near the borders of the vector field were difficult to identify.

5. Conclusions and Future Prospects

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

The problem of dense, 2D, unsteady flow visualization is close to being solved [79]. And with recent follow-up work [38,80], unsteady flow visualization on surfaces is not far behind. However, the generalization to 3D flow fields is still unsolved, especially in the case of unsteady flow. Hardware, arguably, will not be the primary bottleneck to solving this challenge, but perceptual issues will. Perceiving three spatial and three data dimensions directly is a difficult job for the human eye and brain. So far, techniques based on geometric objects and particle animation generalize better to 3D fields.

The scale of numerical flow simulations, and thus the size of the resulting datasets, continues to grow rapidly—generally faster than the size of computer memory. For these reasons more simplification strategies must be conceived, such as spatial selection (slicing, regions of interest), geometry simplification and feature extraction.

Slicing in a 3D field reduces the problem to 2D, allowing use of good 2D techniques, but care must be taken with interpretation, as the loss of the third dimension may lead to physically irrelevant results and wrong interpretation. Taking a single 3D time slice from a 3D time-dependent dataset has similar dangers. Other spatial selections such as 3D region-of-interest selection are less risky, but may lead to loss of context. Reduction of data dimension, such as reducing vector quantities to scalars will give more freedom of choice in visualization techniques (such as 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 datasets and interactive exploration, enabling users to trade accuracy with response time. Some areas that need additional work are:

- dense visualization techniques in 3D,
- multifield visualization with scalar, vector and tensor data,
- handling and exploring huge time-dependent flow datasets,
- user studies for evaluation, validation and field testing of flow visualization techniques,
- visualization of inaccuracy and uncertainty [42,60],
- more robust feature extraction techniques, especially in the case of 3D flow.

We also note that much of the research literature presented here demonstrates methods operating on structured, uniform resolution grids. However, the grids used in the private, commercial industry sector are often adaptive resolution and unstructured, especially in the case of CFD [37,38]. Thus, further research is necessary in order to integrate many of the these methods into practical industrial applications.

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 colleagues from the research community who permitted to use the images as shown in the paper. CFD simulation data used in Figures 2, 6, 9 and 14 courtesy of AVL.

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", and by the Landesstiftung Baden-Württemberg within the "Eliteförderprogramm für Postdoktoranden".

For supplementary material, including additional, higher resolution images and animations, please visit: `http://www.vrvis.at/ar3/pr2/star/`

References

1. D. Arrowsmith and C. Place. *An Introduction to Dynamical Systems*. Cambridge University Press, USA, 1990.
2. H. Battke, D. Stalling and H. Hege. Fast Line Integral Convolution for Arbitrary Surfaces in 3D. In *Visualization and Mathematics*, Springer-Verlag, Heidelberg, pp. 181–195. 1997.
3. J. Becker and M. Rumpf. Visualization of Time-Dependent Velocity Fields by Texture Transport. In *Visualization in Scientific Computing '98*, Eurographics, pp. 91–102, 1998.
4. S. Berger and E. Gröller. Color-Table Animation of Fast Oriented Line Intgral Convolution for Vector Field Visualization. In *WSCG 2000 Conference Proceedings*, pp. 4–11, 2000.

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

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

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

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

9. R. A. Crawfis and N. Max. Texture Splats for 3D Scalar and Vector Field Visualization. In *Proceedings IEEE Visualization '93*, IEEE Computer Society, pp. 261–267, Oct. 1993.

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

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

12. W. de Leeuw and J. van Wijk. Enhanced Spot Noise for Vector Field Visualization. In *Proceedings IEEE Visualization '95*, IEEE Computer Society, pp. 233–239, Oct. 1995.

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

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

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

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

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

18. U. Diewald, T. Preußer and M. Rumpf. Anisotropic Diffusion in Vector Field Visualization onEuclidean Domains and Surfaces. In *IEEE Transactions on Visualization and Computer Graphics*, 6(2):139–149, 2000.

19. G. Erlebacher, B. Jobard and D. Weiskopf. Flow Textures. In C. R. Johnson and C. D. Hansen (eds), *The Visualization Handbook*. Academic Press, Oct. 2003.

20. L. K. Forssell. Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution. In *Proceedings IEEE Visualization '94*, IEEE Computer Society, pp. 240–247, Oct. 1994.

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

22. H. Hege and D. Stalling. Fast LIC with Piecewise Polynomial Filter Kernels. In *Mathematical Visualization*, Springer Verlag, pp. 295–314, 1998.

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

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

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

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

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

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

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

30. B. Jobard and W. Lefer. Creating Evenly–Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97*, volume 7, Eurographics, Springer-Verlag, April 28–30, 1997.

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

32. 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.

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

34. M. Kiu and D. C. Banks. Multi-frequency Noise for LIC. In *Proceedings IEEE Visualization '96*, IEEE, pp. 121–126, Oct. 27–Nov. 1, 1996.

35. D. H. Laidlaw, R. M. Kirby, J. S. Davidson, T. S. Miller, M. da Silva, W. H. Warren and M. Tarr. Quantitative Comparative Evaluation of 2D Vector Field Visualization Methods. In *Proceedings IEEE Visualization 01*, IEEE Computer Society, pp. 143–150, October 2001.

36. D. A. Lane. Scientific Visualization of Large-Scale Unsteady Fluid Flows, *Scientific Visualization: Overviews, Methodologies, and Techniques*. IEEE Computer Science Press, Los Alamitos, chapter 5, pp. 125–145, 1997.

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

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

39. R. S. Laramee, J. Schneider and H. Hauser. Texture-Based Flow Visualization on Isosurfaces from Computational Fluid Dynamics. In *Joint Eurographics—IEEE TVCG Symposium on Visualization (VisSym '04)*, Konstanz, Germany, May 19–21, 2004, forthcoming.

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

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

42. S. K. Lodha, A. Pang, R. E. Sheehan and C. M. Wittenbrink. UFLOW: Visualizing Uncertainty in Fluid Flow. In *Proceedings IEEE Visualization '96*, pp. 249–254, Oct. 27–Nov. 1, 1996.

43. H. Löffelmann, T. Kučera and E. Gröller. Visualizing Poincaré Maps Together with the Underlying Flow. In *Mathematical Visualization*, Springer Verlag, pp. 315–328, 1998.

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

45. H. Löffelmann, Z. Szalavàri and E. Gröller. Local Analysis of Dynamical Systems—Concepts and Interpretation. In *WSCG 1996 Conference Proceedings*, pp. 170–180, Feb. 1996.

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

47. 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*, pp. 57–70. Eurographics, 1997.

48. N. Max and B. Becker. Flow Visualization Using Moving Textures. In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pp. 77–87, Sept. 1995.

49. N. Max, B. Becker and R. Crawfis. Flow Volumes for Interactive Vector Field Visualization. In *Proceedings IEEE Visualization '93*, IEEE Computer Society, pp. 19–24, Oct. 1993.

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

51. A. Okada and D. L. Kao. Enhanced Line Integral Convolution with Flow Feature Detection. In *SPIE Vol. 3017*

Visual Data Exploration and Analysis IV, pp. 206–217, Feb. 1997.

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

53. F. H. Post and T. van Walsum. Fluid flow visualization. In *Focus on Scientific Visualization*, Springer, pp. 1–40, 1993.

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

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

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

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

58. T. Preußer and M. Rumpf. Anisotropic Nonlinear Diffusion in Flow Visualization. In *Proceedings IEEE Visualization '99*, IEEE Computer Society, pp. 325–332, Oct. 1999.

59. C. Rezk-Salama, P. Hastreiter, C. Teitzel and T. Ertl. Interactive exploration of volume line integral convolution based on 3D-texture mapping. In *Proceedings IEEE Visualization '99*, IEEE Computer Society, pp. 233–240, 1999.

60. P. J. Rhodes, R. S. Laramee, R. D. Bergeron and T. M. Sparr. Uncertainty Visualization Methods in Isosurface Rendering. In M. Chover, H. Hagen and D. Tost (eds), *Eurographics 2003, Short Papers*, The Eurographics Association, pp. 83–88, September 1–5, 2003.

61. A. Sanna, B. Montrucchio and P. Montuschi. A survey on visualization of vector fields by texture-based methods. *Recent Research Developments in Pattern Recognition*, 1(1):13–27, 2000.

62. A. Sanna, B. Montrucchio, P. Montuschi and A. Sparavigna. Visualizing Vector Fields: The Thick Oriented Stream-Line Algorithm (TOSL). *Computers and Graphics*, 25(5):847–855, Oct. 2001.

63. A. Sanna, C. Zunino, B. Montucchio and P. Montuschi. Adding a Scalar Value to Texture-Based Vector Field Representations by local contrast analysis. In *Proceedings of the Joint Eurographics—IEEE TCVG Symposium on Visualizatation (VisSym '02)*, pp. 35–41, 2002.

64. G. Scheuermann, H. Burbach and H. Hagen. Visualizing Planar Vector Fields with Normal Component Using Line Integral Convolution. In *Proceedings IEEE Visualization '99*, IEEE Computer Society, pp. 255–262, 1999.

65. H. Shen and D. L. Kao. A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2), Apr.–June, pp. 98–108, 1998.

66. H. W. Shen, C. R. Johnson and K. L. Ma. Visualizing Vector Fields Using Line Integral Convolution and Dye Advection. In *1996 Volume Visualization Symposium*, IEEE, pp. 63–70, Oct. 1996.

67. H. W. Shen and D. L. Kao. UFLIC: A Line Integral Convolution Algorithm for Visualizing Unsteady Flows. In *Proceedings IEEE Visualization '97*, IEEE Computer Society, pp. 317–323, 1997.

68. D. Silver, F. Post, and I. Sadarjoen. *Flow Visualization*, volume 7 of *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, New York, pp. 640–652, 1999.

69. D. Stalling. LIC on Surfaces. In *Texture Synthesis with Line Integral Convolution*, ACM SIGGRAPH 97, International Conference on Computer Graphics and Interactive Techniques, pp. 51–64, 1997.

70. D. Stalling and H. Hege. Fast and Resolution Independent Line Integral Convolution. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, ACM SIGGRAPH, ACM Press/ACM SIGGRAPH, pp. 249–256, 1995.

71. A. Sundquist. Dynamic Line Iintegral Convolution for Visualizing Streamline Evolution. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):273–282, 2003.

72. F. Taponecco and M. Alexa. Vector Field Visualization using Markov Random Field Texture Synthesis. In *Proceedings of the Joint Eurographics–IEEE TCVG Symposium on Visualizatation (VisSym '03)*, Springer-Verlag, pp. 195–202, May 2003.

73. C. Teitzel, R. Grosso and T. Ertl. Line Integral Convolution on Triangulated Surfaces. In *WSCG 1997 Conference Proceedings*, pp. 572–581, 1997.

74. A. Telea and J. van Wijk. Simplified Representation of Vector Fields. In *Proceedings IEEE Visualization '99*, IEEE Computer Society, pp. 35–42, 1999.

75. A. Telea and J. J. van Wijk. 3D IBFV: Hardware-Accelerated 3D Flow Visualization. In *Proceedings IEEE Visualization '03*, IEEE Computer Society, pp. 233–240, 2003.

76. T. Urness, V. Interrante, I. Marusic, E. Longmire, and B. Ganapathisubramani. Effectively Visualizing Multi-Valued Flow Data using Color and Texture. In *Proceedings IEEE Visualization '03*, IEEE Computer Society, pp. 115–122, 2003.

77. M. van Dyke. *An Album of Fluid Motion*. The Parabolic Press, 1982.

78. J. J. van Wijk. Spot noise-Texture Synthesis for Data Visualization. InT. W. Sederberg (ed), *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, ACM, volume 25, pp. 309–318, 1991.

79. J. J. van Wijk. Image Based Flow Visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

80. J. J. van Wijk. Image Based Flow Visualization for Curved Surfaces. In *Proceedings IEEE Visualization '03*, IEEE Computer Society, pp. 123–130, 2003.

81. V. Verma, D. Kao and A. Pang. PLIC: Bridging the Gap Between Streamlines and LIC. In *Proceedings IEEE Visualization '99*, N.Y., pp. 341–348, Oct. 25–29, 1999.

82. H. K. Versteeg and W. Malalasekera. Addison-Wesley, February 1996.

83. R. Wegenkittl and E. Gröller. Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet. In *Proceedings IEEE Visualization '97*, IEEE Computer Society, pp. 309–316, Oct. 19–24, 1997.

84. R. Wegenkittl, E. Gröller and W. Purgathofer. Animating Flow Fields: Rendering of Oriented Line Integral Convolution. In *Computer Animation '97 Proceedings*, IEEE Computer Society, pp. 15–21, June 1997.

85. D. Weiskopf, G. Erlebacher and T. Ertl. A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields. In *Proceedings IEEE Visualization '03*, IEEE Computer Society, pp. 107–114, 2003.

86. D. Weiskopf, G. Erlebacher, M. Hopf and T. Ertl. Hardware-Accelerated Lagrangian-Eulerian Texture Advection for 2D Flow Visualizations. In *Proceedings of the Vision Modeling and Visualization Conference 2002 (VMV-01)*, pp. 439–446, Nov. 21–23, 2002.

87. D. Weiskopf and T. Ertl. GPU-Based 3D Texture Advection for the Visualization of Unsteady Flow Fields. In *WSCG 2004 Conference Proceedings, Short Papers*, pp. 259–266, February 2004.

88. D. Weiskopf, M. Hopf and T. Ertl. Hardware-Accelerated Visualization of Time-Varying 2D and 3D Vector Fields by Texture Advection via Programmable Per-Pixel Operations. In *Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV 01)*, pp. 439–446, Nov. 21–23, 2001.

89. R. Yagel, D. M. Reed, A. Law, P. Shih and N. Shareef. Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing. In *Proceedings 1996 Symposium on Volume Visualization*, pp. 55–62, Sept. 1996.

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