Particle and Texture Based Spatiotemporal Visualization of Time-Dependent Vector Fields

Daniel Weiskopf*[1,2] Frederik Schramm[2] Gordon Erlebacher[3] Thomas Ertl[2]

[1]Graphics, Usability, and Visualization (GrUVi) Lab, Simon Fraser University
[2]Institute of Visualization and Interactive Systems, University of Stuttgart
[3]School of Computational Science and Information Technology, Florida State University

Abstract

We propose a hybrid particle and texture based approach for the visualization of time-dependent vector fields. The underlying spacetime framework builds a dense vector field representation in a two-step process: 1) particle-based forward integration of trajectories in spacetime for temporal coherence, and 2) texture-based convolution along another set of paths through the spacetime for spatially correlated patterns. Particle density is controlled by stochastically injecting and removing particles, taking into account the divergence of the vector field. Alternatively, a uniform density can be maintained by placing exactly one particle in each cell of a uniform grid, which leads to particle-in-cell forward advection. Moreover, we discuss strategies of previous visualization methods for unsteady flow and show how they address issues of spatiotemporal coherence and dense visual representations. We demonstrate how our framework is capable of realizing several of these strategies. Finally, we present an efficient GPU implementation that facilitates an interactive visualization of unsteady 2D flow on Shader Model 3 compliant graphics hardware.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Unsteady flow visualization, visualization framework, LIC, texture advection, particle systems, GPU methods

1 Introduction

Many vector field visualization techniques compute the motion of massless particles along the vector field to obtain characteristic structures like streamlines. With a dense representation like Line Integral Convolution (LIC) [1], the domain is densely covered with particle lines to overcome the issue of appropriately choosing seed points for particle tracing.

In this paper, we focus on the dense visualization of time-dependent vector fields. Any dense representation has to address the challenging goal of achieving spatiotemporal coherence. In Section 4, we analyze the strategies followed by previous methods to construct a spatiotemporally coherent evolution of dense line-like visual patterns, such as streamlines, pathlines, or streaklines.

A generic spacetime framework, as developed in our previous work [3, 27], is one possible approach for the visualization of time-dependent vector fields. This framework builds upon the fact that two types of coherence are important in animated visualizations: spatial coherence, which conveys the structure of a vector field within a single picture by visual patterns, and frame-to-frame coherence, which allows the user to identify the motion of these patterns. The spacetime framework provides explicit control over both types of coherence by means of a two-step process. The first step generates continuous trajectories in spacetime to achieve temporal coherence. The second step generates spatial patterns by convolution along another set of paths through the spacetime volume.

In this paper, we adopt our previous texture-based implementation of the framework [27] and improve the construction of spacetime trajectories by replacing texture-based backward gathering with particle-based forward integration. The main advantage of forward integration is a significant reduction of computations and memory footprint. A typical speedup is in the range of 10–100 for a comparable visualization quality. In particular, the particle approach is extremely efficient when the density of the representation is low. Another advantage is that constant spatial frequencies can be maintained. We describe two alternative methods to achieve a uniform density of particles over time: The first approach controls particle density by stochastic injection and removal of particles. The second approach checks and adjusts the particle density that is explicitly measured on cells of a uniform grid. In a special case, exactly one particle is kept per cell, leading to the new *particle-in-cell* advection method.

The combined particle/texture-based framework retains all important advantages of the previous, purely texture-based implementation [27]. First, its high degree of flexibility allows us to implement or mimic previous flow visualization methods. Second, high visualization quality is achieved by Lagrangian integration, which avoids artificial blurring inherent to Eulerian or semi-Eulerian methods. Third, a direct mapping to graphics processing units (GPUs) leads to an efficient implementation. The GPU implementation of the particle-based method makes use of a coupling between texture information and vertex processing, which is facilitated by GPUs with Shader Model 3 support.

2 Previous Work

The discussion of previous work focuses on noise-based and dense representations [10]. Early texture-synthesis techniques for vector field visualization are spot noise [22] and LIC [1]. Although LIC has been extended in various respects, specific methods for time-dependent vector fields and animation are particularly interesting in the context of this paper, such as Unsteady Flow LIC (UFLIC) [19], Accelerated UFLIC (AUFLIC) [14], Dynamic LIC (DLIC) [20], and Unsteady Flow Advection–Convolution (UFAC) [27].

A related class of dense representations makes use of texture advection, which represents a dense collection of particles in a texture and transports this texture along the vector field [15]. Both Lagrangian-Eulerian Advection (LEA) [5] and Image-Based Flow Visualization (IBFV) [23] adopt the idea of texture advection to achieve an interactive visualization of 2D unsteady flow. Recent progress in interactive flow visualization has been driven by the development of fast and flexible GPUs [4, 23, 29]. GPU-based interactive visualization can be extended to vector fields on curved surfaces [11, 24, 28] and in 3D [21, 29, 30]. The Chameleon system [13] allows the user to interactively modify the rendering style

*weiskopf@vis.uni-stuttgart.de

IEEE Visualization 2005
October 23-28, Minneapolis, MN, USA
0-7803-9462-3/05/$20.00 ©2005 IEEE.

of pre-computed particle paths. Efficient particle tracing can be accomplished when a vector field is used to drive the evolution of a GPU-based particle system [7, 8, 9]. Finally, frame-to-frame coherence can be achieved for the animation of geometrically constructed streamlines [6, 12].

3 Nomenclature

This section describes the nomenclature used for vector fields and different types of particle traces throughout this paper. The terminology is adopted from our previous discussion [27] and allows us to explicitly distinguish between spatial and temporal properties.

In Euclidean n-dimensional space, a time-dependent vector field is a map $\mathbf{u}(\mathbf{x},t)$ that assigns a vector to each point $\mathbf{x}$ in space at time t. In what follows, lower-case boldface letters denote vectors or points in nD space $\mathbb{R}^n$. Pathlines $\mathbf{x}_{\text{path}}$ of the vector field are governed by the ordinary differential equation

$$\frac{\mathrm{d}\mathbf{x}_{\text{path}}(t;\mathbf{x}_0,t_0)}{\mathrm{d}t} = \mathbf{u}(\mathbf{x}_{\text{path}}(t;\mathbf{x}_0,t_0),t) \ , \qquad (1)$$

with the initial condition $\mathbf{x}_{\text{path}}(t_0;\mathbf{x}_0,t_0) = \mathbf{x}_0$. In general, we adopt a notation in which $\mathbf{x}(t;\mathbf{x}_0,t_0)$ describes a curve parameterized by t that yields the point $\mathbf{x}_0$ for $t = t_0$.

Particle motion can be investigated in spacetime, i.e., in a manifold with one temporal and n spatial dimensions. The world line—the spacetime curve—traced out by a particle can be written as $\mathscr{Y}(t;\mathbf{x}_0,t_0) = (\mathbf{x}_{\text{path}}(t;\mathbf{x}_0,t_0),t)$. In general, curves in spacetime are denoted by scripted variables and have $n+1$ components: n spatial and one temporal. $\mathscr{Y}(t;\mathbf{x}_0,t_0)$ is parameterized by its first argument and passes through the point $(\mathbf{x}_0,t_0)$ when $t = t_0$. We use the term trajectory for the spacetime description of a pathline, i.e., a trajectory can be considered as a pathline that is lifted from nD space to $(n+1)$D spacetime.

Besides pathlines, there exist two important additional types of characteristic curves for a time-dependent vector field: streamlines and streaklines. A streamline is defined as the particle path associated with an artificially steady, instantaneous vector field at a fixed physical time τ, which is governed by

$$\frac{\mathrm{d}\mathbf{x}_{\text{stream}}(t;\mathbf{x}_0,t_0)}{\mathrm{d}t} = \mathbf{u}(\mathbf{x}_{\text{stream}}(t;\mathbf{x}_0,t_0),\tau) \ .$$

Here, t and t_0 are just parameters along the curve and do not have the meaning of physical time.

A streakline is produced by dye that is continuously released into a vector field. To obtain the snapshot of a streakline at time t, particles are released from $\mathbf{x}_0$ at times $s \in [t_{\min},t]$ and their positions are evaluated at time t: $\mathbf{x}_{\text{streak}}(s;\mathbf{x}_0,t) = \mathbf{x}_{\text{path}}(t;\mathbf{x}_0,s)$. The streakline is parameterized by s, and $t_{\min}$ is the first time that particles are released.

4 Strategies for Unsteady Flow Visualization

Characteristic curves of a flow are good candidates to build dense line-like vector field representations. In fact, pathlines, streamlines, and streaklines are employed in various visualization methods for time-dependent data sets. A fundamental goal of any visualization method is to construct a spatiotemporally coherent visualization. This goal is not trivially achieved; for example, as discussed by Shen and Kao [19], a straightforward application of pathlines and streamlines leads to problems with spatial and temporal correlations in a dense representation. A related issue is that pathlines (or streaklines) may intersect each other, which makes them a problematic choice for a consistent representation.

We discuss several different strategies used in previous work to achieve a useful visualization of unsteady flow. The first strategy lifts the visualization from nD space to $(n+1)$D spacetime. This higher-dimensional description avoids the problem of intersecting lines because spacetime trajectories are unique and cannot intersect. The disadvantage of this approach is that the problems are essentially postponed to a later stage of visualization when a final image has to be rendered on a 2D display [31].

The second strategy is to replace characteristic curves by arbitrary curves. A generic curve transported along a vector field is called a timeline. Timelines have the advantage of being temporally coherent, and they do not intersect each other at any time if they do not intersect each other at seed time. However, timelines usually become convoluted after a short advection period. Another problem is that timelines have no direct relationship to the vector field; they do not show direction or magnitude of the flow. Therefore, dense timelines are rarely used.

A third strategy is to relax the goal of constructing dense *line-like* structures. LEA [5] and IBFV [23], for example, are based on line integral convolution along (time-reversed) streaklines. Since streaklines may intersect, this strategy leads to visual patterns that are smeared out in more than just a single direction. Similarly, UFLIC [19] and AUFLIC [14] generate widened "lines" for unsteady vector fields. Therefore, LEA, IBFV, UFLIC, and AUFLIC achieve a temporally coherent visualization at the cost of a non-constant resolution of spatial structures. The extent of line widening can be decreased by using short curves, e.g., streaklets or pathlets, which reduce the chance of overlapping and intersecting lines. A sparse representation, e.g., based on Oriented LIC (OLIC) [25] or dye injection at a few isolated locations, can also be used to reduce the problem of overlapping curves because a few isolated lines can be distinguished even when they intersect.

The fourth strategy is to relax temporal coherence. A straightforward approach builds an animation from a collection of LIC computations for different time steps. A naive implementation of this method leads to significant flickering. UFAC [27], however, reduces or even removes flickering by controlling the length of streamlines as a function of unsteadiness.

The fifth strategy is to transport line-like visual patterns not along the original vector field, but to choose an evolution along a different vector field. As demonstrated by DLIC [20], this approach is useful for vector fields that do not represent a physical motion of material, e.g., for time-dependent electric fields.

The third strategy is most popular for the visualization of unsteady flow when the range of research and applications is considered. Although previous research addresses issues of temporal and spatial coherence, none of the papers on LEA, IBFV, UFLIC, or AUFLIC describes explicitly what kind of line-like patterns are shown in a single snapshot of an animation. In our previous work [3], we have attempted to identify the types of lines generated by these visualization methods because we believe that the characteristics of different methods can be best assessed with an explicit specification of both the temporal and spatial properties.

The following section describes a visualization framework suitable for the third, fourth, and fifth of the aforementioned strategies.

5 Continuous Framework

We briefly discuss a generic continuous framework that targets an explicit model of a spatiotemporal visualization. The framework is detailed in our previous work [3, 27].

The basic idea of the framework is to adopt a spacetime view on particle tracing. In general, a dense representation of a vector field employs a large number of particles so that the intersection between each spatial slice and the trajectories yields a dense coverage by points. Accordingly, spacetime itself is densely filled by these trajectories $\mathscr{Y}(s;\mathbf{x},t)$, associated with the vector field $\mathbf{u}(\mathbf{x},t)$. Properties that typically comprise just a gray-scale value from the range $[0,1]$ can be attached to particles to distinguish them from one another. Properties are allowed to change continuously along the trajectory; very often, however, they remain constant. From a collection of trajectories, a property function $I(\mathbf{x},t)$ can be defined by identifying its value with the property of the particle that crosses through the spacetime location $(\mathbf{x},t)$. The function value is set to zero if the location is not covered by a trajectory. By this construction, $I(\mathbf{x},t)$ fills the complete spacetime domain. The continuous behavior of trajectories and their attached properties guarantees that spatial slices through the property field $I(\mathbf{x},t)$ at nearby times are strongly related, i.e., these slices are temporally coherent. An animated sequence built from spatial slices with increasing time results in the motion of particles governed by the vector field.

Since, in general, different particles are not correlated, spatial slices of the property field do not exhibit any coherent spatial structures. To achieve spatial correlation, a filtered spatial slice $D_t(\mathbf{x})$ is defined through the convolution

$$D_t(\mathbf{x}) = \int_{-\infty}^{\infty} k(s) I(\mathscr{Z}(t-s;\mathbf{x},t))\, \mathrm{d}s \qquad (2)$$

along $\mathscr{Z}(s;\mathbf{x},t)$. The subscript on D_t is a reminder that the filtered image depends on time. The kernel $k(s)$ need not be the same for all points on the filtered slice and may depend on additional parameters, such as derived vector field data. $\mathscr{Z}(s;\mathbf{x},t)$ can be any path through spacetime and need not be the trajectory of a particle. However, the spatial components of the path are given by the pathlines of some other vector field $\mathbf{w}(\mathbf{x},t)$. The temporal component of $\mathscr{Z}(s;\mathbf{x},t)$ may depend on s and t.

An animated visualization is composed of images D_t for uniformly increasing time t. The structure of D_t is defined by the triplet $[I,\mathscr{Z},k]$, where I is built from trajectories $\mathscr{Y}$. The main advantage of this generic framework is its separate control over the temporal evolution along pathlines of $\mathbf{u}(\mathbf{x},t)$ and over the spatial structures that result from convolution along paths based on $\mathbf{w}(\mathbf{x},t)$.

Several parameter choices for this triplet lead to useful visualizations [27]. For example, the framework-based version of LEA [5] is realized by setting $\mathbf{u}(\mathbf{x},t)$ to the given input vector field $\mathbf{v}(\mathbf{x},t)$, and $\mathbf{w}(\mathbf{x},t) = 0$. LEA constructs a dense collection of time-reversed streaklets, adopting the third strategy from Section 4. Similarly, IBFV [23] is based on a dense collection of streaklets, obtained by using $\mathbf{u}(\mathbf{x},t) = 0$ and $\mathbf{w}(\mathbf{x},t) = \mathbf{v}(\mathbf{x},t)$. In contrast, UFAC [27] sets $\mathbf{u}(\mathbf{x},t)$ to the input vector field $\mathbf{v}(\mathbf{x},t)$ and performs the convolution along instantaneous streamlines at time t' of the same field, i.e., $\mathbf{w}(\mathbf{x},t) = \mathbf{v}(\mathbf{x},t')$ and $\mathscr{Z}(s;\mathbf{x},t') = (\,\cdot\,,t')$. The length of streamlines is controlled by the unsteadiness of the vector field, following the fourth strategy from Section 4. Finally, DLIC [20] targets the animation of instantaneous streamlines (or fieldlines) of a time-dependent vector field. Two different vector fields are used for a framework-based version of DLIC: $\mathbf{w}(\mathbf{x},t)$ governs the streamline generation at each time step, while $\mathbf{u}(\mathbf{x},t)$ describes the evolution of streamlines. DLIC can be considered as an example of the fifth strategy from Section 4, which transports line-like visual patterns along a separate vector field. Therefore, the framework is capable of implementing three important strategies for time-dependent vector fields, through an appropriate choice of parameters.

6 Particle-Based Discretization

A discretization of the continuous framework allows us to compute visualization images. The property field I, the visualization images D_t, and the vector fields are represented by discrete uniform grids—or textures. The implementation can be separated into two major parts: First, spatial slices of the property field I are constructed from trajectories of one vector field; the complete property field is generated on the spacetime domain by combining spatial slices. Second, convolution is performed along $\mathscr{Z}$ within the spacetime property field.

The previous discretization [27] implements both parts by a purely texture-based gathering of property contributions: Starting from a texel, trajectories are traced and potential contributions are gathered and accumulated from locations along these trajectories. Benefits of this implementation are a high-quality visualization due to Lagrangian integration (as opposed to artificial blurring in Eulerian and semi-Lagrangian approaches) and a highly flexible framework that facilitates various visualization methods. Unfortunately, the gathering approach requires a recurring and time-consuming integration of particle traces to construct the spacetime property field I. Each spatial slice through I triggers a complete integration of long trajectories backward in time. The computational costs are determined by the maximum particle lifetime, on the order of 20–200 time steps. A related problem is that a significant temporal portion of the data set needs to be kept in memory because backward particle tracing must access the corresponding time steps of the vector field. A third issue is a non-constant spatial frequency in slices through I, introduced by the divergence of the vector field.

The goal of our new discretization scheme is to overcome these problems and, at the same time, retain the aforementioned benefits. The basic idea is to replace gathering based on backward integration by an approach that propagates particles forward in time. The changes mainly concern the first stage of the framework, i.e., the construction of the spacetime volume.

To achieve a high flexibility in designing the injection of new particles, we support a continuous description of injected "particles" on a spatial slice of constant time t. We use radial basis functions (RBFs) [17, 18], which are spherically symmetric functions around associated center points, to represent the injection of "particles":

$$I_{\text{inject}}(\mathbf{x}) = \sum_i \lambda_i \phi_i(||\mathbf{x}-\mathbf{x}_i||) \ ,$$

with an index i that labels a particle, the corresponding weight λ_i, the center $\mathbf{x}_i$, and the radial basis function $\phi_i(r)$. Common types of basis functions for numerical approximations are thin-plate splines, multiquadrics, inverse multiquadrics, or Gaussians. The goal of this paper is to model particle traces, not to approximate a generic function. Therefore, basis functions with compact or quasi-compact support are adequate because they represent localized particles of finite size. We typically use Gaussian functions, $\phi(r) = \exp(-r^2/(2\sigma^2))$, with width σ.

For the temporal evolution of injected particles, we assume that the center points $\mathbf{x}_i$ travel along pathlines and the basis functions ϕ_i remain constant. However, weights λ_i may change over time to model a phase-in and phase-out of particles. Additional external parameters, such as additional attributes of a flow, may also influence λ_i. For simplicity of notation, these parameters are not explicitly included in the mathematical expressions. The scenario is extended to a repeated injection that occurs at several discrete time levels τ. Then, the spacetime property field is

$$I(\mathbf{x},t) = \sum_{\tau \le t} \sum_i \lambda_{\tau,i}(t-\tau)\phi_{\tau,i}(||\mathbf{x}-\mathbf{x}_{\text{path}}(t;\mathbf{x}_{\tau,i},\tau)||) \ . \qquad (3)$$

The additional subscript τ labels the time of particle injection. Moreover, the weight $\lambda_{\tau,i}$ depends on the age of a particle, $t-\tau$.

This formulation allows us to compute spatial slices of I incrementally: If pathlines have been determined for time t, particle positions at the following time step $t+\Delta t$ can be calculated by integrating the particle tracing Equation (1) only for a small time interval Δt, i.e., a re-computation of complete pathlines is avoided. In this state-preserving approach, current particle positions are stored for later reuse. Particle states are held in an array that represents all active particles. The particle state comprises position $\mathbf{x}$, age $t-\tau$, Boolean state for activity, and possible additional attributes. The activity state allows us to remove a particle from the system by setting its activity state to false. Particle removal is useful to free computational and memory resources for particles that have died, i.e., whose weight is zero for all future times. Therefore, the size of the particle array can be restricted to the maximum number of simultaneously active particles. A new particle is injected by filling a previously inactive array element with the initial particle position, and by setting the age to zero and the activation state to true. The particle system is propagated from time t to $t+\Delta t$ by updating positions according to a Lagrangian integration of Eq. (1) and by adding Δt to the age of a particle.

The construction of the property field in Eq. (3) requires the evaluation of RBFs in a finite region around particle positions, defined by the size of the support of the RBFs. This process is identical to a scan conversion of RBFs on an nD spatial domain if the property field is discretized on a uniform grid. Then, the summation in Eq. (3) can be computed by additively blending several particles.

The second major part of the discretization implements the convolution along trajectories $\mathscr{X}$ through the spacetime property field. The basic idea is to discretize the convolution integral (2) by a Riemann sum. Since the convolution is performed for each time step separately, this part of the framework generally cannot benefit from incremental computations. Therefore, the calculations are performed for each time step and each texel independently.

The incremental construction of the spacetime property field has several advantages. First, pathlines are computed incrementally and, thus, unnecessary re-computations of complete trajectories are avoided. Second, the computation time is linear in the number of particles, i.e., the particle-based method is very efficient for moderately sparse representations like OLIC [25]. Third, the memory footprint for the time-dependent vector field $\mathbf{u}(\mathbf{x},t)$ is significantly cut down to just a single time step. Fourth, for the case of LEA-type visualization within the framework, the convolution stage can be efficiently realized by a repeated alpha blending of subsequent slices through the spacetime volume [27], i.e., the convolution computation can also be performed incrementally. Therefore, LEA-style visualization nicely fits in with the incremental construction of the property field.

7 Probabilistic Density Control

A problem of any particle-oriented visualization method is an effective control of particle density. For example, a divergent flow makes particles drift away from each other and, thus, reduces particle density. Our goal is to control particle density while avoiding its explicit evaluation. To this end, a probabilistic approach is applied: We assume that a large number of particles covers the domain so that the law of large numbers, as formalized by Chebyshev's inequality, can be used to identify the density of discrete particles with a continuous material density. The idea is to specify probabilities for the injection of new particles and the removal of old particles in a way which ensures that an imposed particle density is maintained.

We begin the discussion with a continuous description in 3D and later apply it to a discretized particle formulation and to the 2D case. A continuous stream of material is characterized by its velocity $\mathbf{u}$ and mass density ρ, which can be combined to form the density of mass flow, $\mathbf{j} = \rho\mathbf{u}$. The total mass in a volume V is $M = \int_V \rho(\mathbf{x},t)\,\mathrm{d}^3x$. Mass can be changed by inflow and outflow of material according to $\mathrm{d}M/\mathrm{d}t = -\oint_{\partial V} \mathbf{j}(\mathbf{x},t)\cdot \mathrm{d}\mathbf{A}$, where ∂V and $\mathrm{d}\mathbf{A}$ describe the boundary of the volume and an oriented surface element of the boundary, respectively. Assuming conservation of mass, a change of mass density is exclusively caused by inflow and outflow:

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \int_V \frac{\partial \rho(\mathbf{x},t)}{\partial t}\,\mathrm{d}^3x = -\oint_{\partial V} \mathbf{j}(\mathbf{x},t)\cdot \mathrm{d}\mathbf{A} = -\int_V \nabla\cdot\mathbf{j}(\mathbf{x},t)\,\mathrm{d}^3x \ . \tag{4}$$

The last equality is due to the general Stokes' theorem, valid in nD space. The differential form of Eq. (4) yields the continuity equation

$$\nabla\cdot\mathbf{j}(\mathbf{x},t) + \frac{\partial \rho(\mathbf{x},t)}{\partial t} = 0 \ .$$

Density can be controlled by introducing new material or removing existing material, i.e., the continuity equation needs to be extended to

$$\sigma_{\mathrm{inj}}(\mathbf{x},t) - \rho(\mathbf{x},t)\sigma_{\mathrm{del}}(\mathbf{x},t) = \nabla\cdot(\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)) + \frac{\partial \rho(\mathbf{x},t)}{\partial t} \ , \tag{5}$$

where σ_{inj} describes the rate of injection of mass density per time interval and σ_{del} the relative rate of mass removal. Equation (5) can be simplified by taking into account two useful approximations. First, $\nabla\cdot(\rho\mathbf{u}) \approx \rho\nabla\cdot\mathbf{u}$ if a slow spatial change of the density ρ is assumed. Second, if the time scales for changes of ρ are larger than the time scales for injection and removal of particles, the term $\partial\rho/\partial t$ can be neglected because the particle system is in a dynamic equilibrium. Similarly to the treatment in equilibrium thermodynamics, we still can change ρ over time, with a brief delay caused by tuning-in. However, in most cases, we would like to maintain a temporally and spatially constant density anyway. The two approximations lead to

$$\sigma_{\mathrm{inj}}(\mathbf{x},t) = \rho(\mathbf{x},t)(\sigma_{\mathrm{del}}(\mathbf{x},t) + \nabla\cdot\mathbf{u}(\mathbf{x},t)) \ . \tag{6}$$

The choices for σ_{inj} and σ_{del} are not independent. In addition, σ_{inj} and σ_{del} are constrained to non-negative values because an "inverse" injection or removal of particles is impossible. One possibility is to set σ_{inj} to a globally fixed, user-defined value. Then, σ_{del} is used to locally control material density. The removal rate is proportional to the amount of existing material ρ. From a probabilistic point of view, $(\sigma_{\mathrm{del}}\Delta t)$ is a measure for the probability that a particle is removed during time step Δt, i.e., particle removal can be implemented by Russian roulette. If a vector field $\mathbf{u}$, an arbitrary but fixed value σ_{inj}, and a density distribution ρ are given, the corresponding σ_{del} can be computed.

An alternative approach sets σ_{del} to a globally fixed value and determines an associated space-dependent σ_{inj}. The injection density σ_{inj} can be related to a corresponding probability density by normalization: $\sigma^*_{\mathrm{inj}}(\mathbf{x},t) = \sigma_{\mathrm{inj}}(\mathbf{x},t)/\int_V \sigma_{\mathrm{inj}}(\mathbf{x},t)\,\mathrm{d}^3x$. Therefore, injection can be modeled by probabilistically generating new particles according to this probability density. A third approach uses a space-dependent injection and removal of particles, combining the other two methods. In all three approaches, density can be globally scaled with a constant factor ξ by uniformly scaling σ_{inj} with the same factor ξ. Therefore, the density or sparseness of the representation can be easily adjusted by modifying the injection rate, i.e., the number of new particles per unit time.

Even when the expectation value for particle density is kept constant, a modification of injection and removal rates may affect temporal coherence. For example, low injection and removal rates lead to high temporal coherence. Conversely, high injection and removal rates reduce the time and length scales over which the equilibrium state is achieved: They are better for quickly adapting density—at the cost of decreased temporal coherence.

A probabilistic injection of particles requires the construction of associated probability density functions. We assume that a (pseudo) random number generator provides a uniform random variable. From a 1D uniform distribution, a non-uniform nD probability density function can be constructed in various ways [2]. We use rejection sampling to construct the probability density σ^*_{inj}. Russian

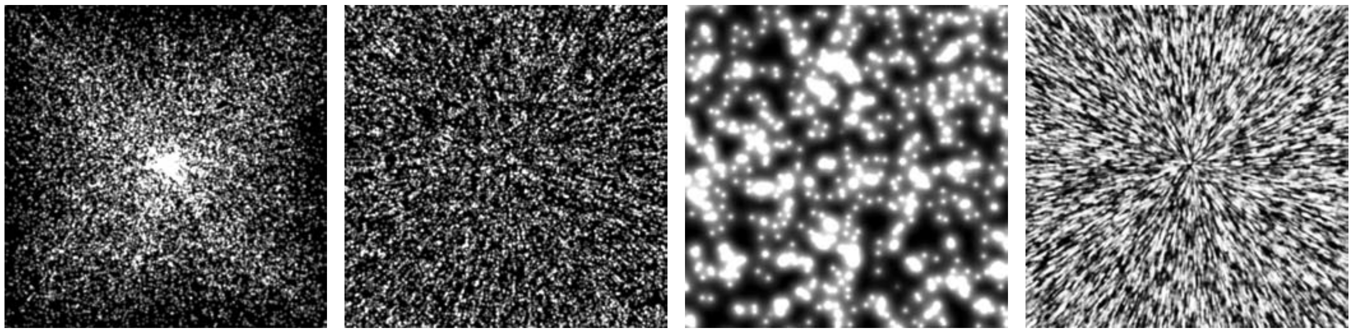

(a) (b) (c) (d)

Figure 1: Probabilistic density control for a flow with a sink in the center. Image (a) shows the particle distribution without density control, (b) with density control, (c) a sparse representation with density control, and (d) the particle system from (b) with LEA-type alpha blending.

roulette for particle removal can be achieved with the inversion method: A single, uniformly distributed random number is drawn; if the number is less than the probability for particle deletion, the particle is removed. The discussion of this section can be immediately transferred from 3D domains to 2D domains because Stokes' theorem also applies in 2D. The only difference is that boldface variables represent 2D vectors instead of 3D vectors and that volume V is replaced by area A.

Figure 1 illustrates the probabilistic density control for a 2D vector field with radial inflow and a sink at the center. The flow can be best recognized in the LEA-type visualization displayed in Figure 1d. Figures 1a–c show snapshots of the particle system, without the second convolution stage of the framework. Figure 1a illustrates a spatially uniform injection and removal of particles. Due to convergence of the flow, particles clump together in the center of the image. Figure 1b demonstrates probabilistic density control by adapting both injection and removal properties to the divergence of the flow, which achieves a uniform density. Figure 1c shows another example of density control, with a much smaller density and larger RBFs than in Figure 1b. Animations corresponding to the four images can be found on our project web page [26].

8 Explicit Density Control

An alternative approach for density control explicitly checks the density of particles. We compute the density by counting particles in cells of a uniform grid covering the domain. After each particle integration step, the cell-based density is computed and compared with the given density. If both differ by more than a user-defined tolerance threshold, surplus particles have to be removed or, alternatively, missing particles have to be injected.

We propose an efficient algorithm for a special case of deterministic density control—for a uniform density with exactly one particle per cell. Due to this particle-per-cell property we name this method *particle-in-cell* advection. The approach is also based on the particle framework from Section 6; it just uses a different method for density control than Section 7.

The particle-in-cell technique consists of the following stages. First, a valid particle configuration is initialized: One particle is chosen per cell, with randomized fractional coordinates. The cell contents is identified with the particle property. Initial properties are modeled as white noise. Second, particles are transported according to forward Lagrangian integration. Third, a re-initialization is performed after each integration step to re-establish a valid configuration. This means that maximally one particle is stored per cell. If several particles are transported to the same cell, only one of them is kept, which can be realized by overwriting existing particles by new ones. A valid configuration also means that minimally one particle has to be located in each cell. Since forward transport may leave destination cells untouched, all cells have to be checked if they are empty, and empty cells have to be filled with a new particle. The property of a new particle is set to a random value.

Figure 2 illustrates particle-in-cell advection based on a circular vector field. Figure 2a (left part) shows a snapshot of the particle distribution from an animation. This image demonstrates that a random and uniform collection of particles is maintained. The right part of Figure 2a uses yellow color to visualize the empty cells that occur after one particle integration step. This picture indicates that a significant portion of the cells may receive new particles, which reduces temporal coherence. Figure 2b shows a LEA-type visualization—with a continuous alpha blending between subsequent particle distributions—leading to streaklets. Here, LIC post-filtering [5] with a short filter kernel is additionally applied to increase the visualization quality. In general, particle-in-cell advection provides a rather low temporal and spatial coherence due to a high rate of particle creation and deletion. Therefore, the probabilistic approach from Section 7 is more appropriate when high quality is required.

Particle-in-cell advection is related to the original implementation of Lagrangian Eulerian Advection (LEA) [5]. LEA also maintains a single particle per cell, it uses Lagrangian integration, and a re-initialization step. However, there are some differences: First, LEA applies backward integration and backward texture mapping. Second, LEA suffers from "noise duplication" [5], which reduces the spatial frequency of the noise. Therefore, LEA needs a continuous ad-hoc injection of noise, which is not required for particle-in-cell advection.

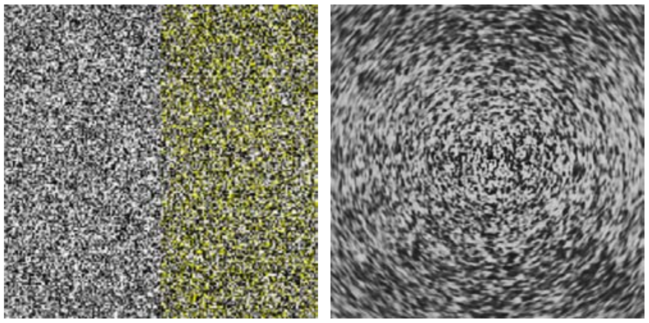

(a) (b)

Figure 2: Particle-in-cell advection for a circular flow. Image (a) shows a snapshot of the particle distribution (left part), indicating empty cells by yellow color (right part). Image (b) displays the result of temporal blending in combination with LIC post-filtering.

9 Implementation

We have implemented the framework from Section 6, including probabilistic and particle-in-cell density control for time-dependent 2D vector fields. The implementation is based on C++ and DirectX 9.0. GPU states, vertex programs, and fragment programs are configured within effect files, using HLSL (high-level shading language) for the shader programs. The implementation requires Shader Model 3 functionality and was developed on NVIDIA GeForce 6 GPUs. The implementation with probabilistic density control can be separated in four major parts: (1) Lagrangian particle representation and integration, (2) probabilistic density control, (3) construction of spacetime property fields, and (4) convolution. Particle-in-cell advection adopts this pipeline and modifies it by an alternative control of particle density.

The first part of the probability-based pipeline is implemented similarly to previous GPU particle systems [7, 8, 9]. We use a state-preserving particle system that holds the current position (its x and y components), age, and activity state for each particle. Particle states are stored in 32-bit floating-point 2D textures, labeled by particle IDs. Particle positions for a subsequent time step are determined by explicit Euler integration of the particle tracing Equation (1), based on a vector field stored in a 16-bit floating-point 2D texture. The integration step is implemented by updating the state texture in a texel-by-texel fashion within a fragment program. Here, render-to-texture functionality is employed in combination with ping-pong rendering, i.e., two versions of a texture are held on the GPU, one serving as render target, the other one serving as lookup texture. Both textures are exchanged after each rendering pass. The integration step also updates the particle's age and checks whether the particle has left the computational domain. In the latter case, the particle is deactivated.

The second part implements particle removal and injection in order to control density. Random numbers are pre-computed by the CPU and transferred to a texture on the GPU. Particle removal needs just a single uniformly distributed random number per particle. Russian roulette is implemented by comparing the entry from the random number texture with a probability value accessed from a separate 2D texture that holds the divergence of the vector field, pre-computed on the CPU. Particle injection is realized by another random number texture, constructed from an injection probability density by rejection sampling on the CPU. This texture yields the random 2D position of a new particle. Particle injection can be performed for previously inactive state-texture elements. Particle removal and injection is implemented within a fragment program that processes all particles in a texel-by-texel fashion via ping-pong rendering. Both random textures can be reused for several time steps by applying a randomized overall rotation of texture coordinates.

In the third part, a spacetime property field is constructed from particle trajectories according to Eq. (3). RBFs are evaluated by rasterizing point sprites defined by a center point (i.e., the particle position) and a 2D texture (i.e., a 2D discretized version of the RBF). A phase-in and phase-out of particles is employed to avoid flickering; the particle's age is used to modulate the brightness of the sprite. With Shader Model 3 compliant GPUs, a vertex program can access texture data and, therefore, the geometry pipeline can read the particle-state texture to position the center of a point sprite. The vertex texture is addressed by texture coordinates corresponding to the particle ID. The summation of several RBFs according to Eq. (3) is implemented by an additive blending of the point sprites corresponding to active particles into a common render-target texture. We use a 16-bit floating-point target, which supports alpha blending and leads to higher accuracy than 8-bit fixed-point textures.

The fourth part implements the convolution according to Eq. (2). Similarly to our previous implementation [27], Lagrangian particle integration is applied to perform line integral convolution. The main difference is that the integral (2) is evaluated in a single rendering pass through a GPU fragment program that executes a loop over all sampling positions in the Riemann sum. Shader Model 3 facilitates loops and long fragment programs (in contrast to Pixel Shader 2) and, therefore, multiple rendering passes can be avoided. An alternative convolution model is used for framework-based LEA: Successive alpha blending is applied between the property field for the current time step and the previously filtered property field, realizing a discretized version of an exponential filter kernel [3].

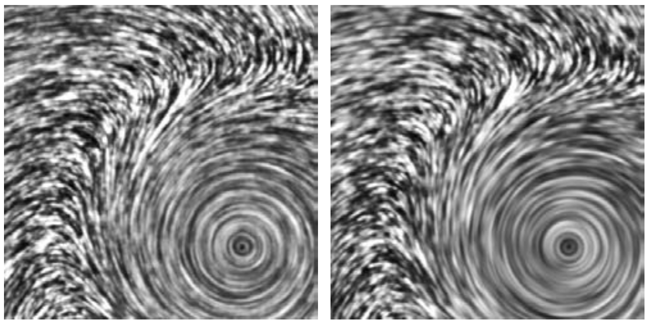

(a) (b)

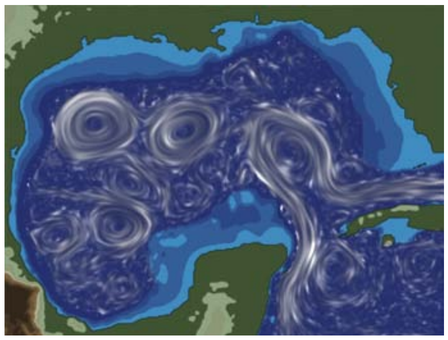
Figure 3: Framework-based LEA-type visualization of a vortical flow. Image (a) shows spatial structures from alpha blending, image (b) adds post-filtering to improve the visualization quality.

Particle-in-cell advection uses a slightly modified pipeline to implement an explicit control of particle density. A valid particle configuration is constructed by the CPU, and particle transport is identical to that of the original pipeline. During re-initialization, a vertex program accesses the particle-state texture to fill a single texel at the new particle position. If several particles are transported to the same texel, they are automatically overwritten by the latest drawn particle, leading to a random selection of surviving particles. A separate rendering pass visits all texels of the particle-state texture and fills empty texels by a random property value and random fractional coordinates. Finally, subsequent time steps of the property field are combined by successive alpha blending.

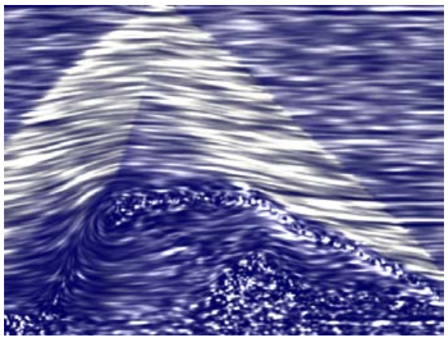
Several postprocessing stages have been implemented to improve visualization quality. First, histogram equalization [16] reestablishes good contrast. Second, LIC post-filtering [5] can be applied to improve filtering quality. The implementation of the convolution stage is reused here, just with a very short filter kernel (typically some five integration steps). Figure 3 illustrates post-filtering with a Gaussian filter kernel, applied to a framework-based LEA-type visualization of a vortical flow. Third, color mapping is used to visualize additional attributes or to enhance flow features by masking, e.g., by velocity masking [5].

10 Results

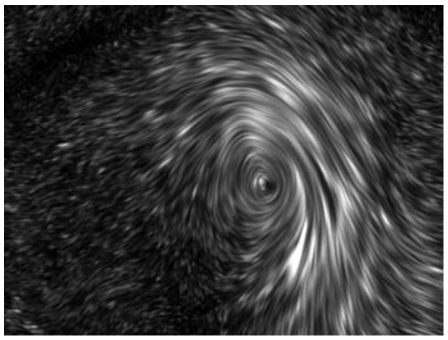
Figure 4 shows snapshots taken from animated visualizations of three different unsteady vector fields. In Figure 4a, a LEA-style framework-based visualization with velocity masking is used to display time-dependent water flow in the Gulf of Mexico. The complete data set contains 183 time steps with a spatial resolution of 352×320. Figure 4b is generated by framework-based UFAC and shows the velocity field produced by the interaction of a planar shock with a longitudinal vortex (200 time steps with a resolution of 256×151). In regions of large unsteadiness, such as the shock regions at center and bottom of the image, the correlation length along streamlines is reduced by UFAC. In addition, velocity masking is applied to emphasize high velocity magnitude. In Figure 4c, a



(a)



(b)



(c)

Figure 4: Examples for unsteady flow visualization: (a) LEA-style framework-based visualization of water flow in the Gulf of Mexico. (b) UFAC visualization of the interaction of a planar shock with a longitudinal vortex. (c) LEA-style visualization of a hurricane Dennis prediction.

LEA-style framework-based visualization with velocity masking is applied to represent the wind velocity in a hurricane Dennis prediction for a 96-hour forecast (17 time steps with a spatial resolution of 151×181). Corresponding animations and additional images can be found on our project web page [26].

Other types of results concern the performance of our implementation. All measurements were conducted on a PC with NVIDIA GeForce 6800 Ultra GPU (256 MB) and Intel Pentium IV (3.2 GHz) CPU, running under Windows XP and DirectX 9.0. Viewport size is 640×480 unless otherwise noted. Performance numbers are always given in frames per second (fps). Table 1 shows the performance of the particle system, including point sprite rendering and blending, for a varying number of particles, different sizes of point sprites (sizes on the image plane), and a steady flow. The column "w/o VP" shows performance numbers for particle tracing only, i.e., no point sprites are drawn and there is no texture access from the vertex program to the particle state texture. When point sprites are rendered, the overall performance drops significantly with increasing size of point sprites, which shows that an overdraw from rasterization is the bottleneck of the typical particle system, not particle integration. The column with point sprite size "0^2" documents the cost for a texture lookup in the vertex program without any rasterization of point sprites. For comparison, the gathering implementation for particle transport [27] achieves 11.4 fps for a maximum particle lifetime of 200 on the same GPU, i.e., our new discretization achieves a speedup by a factor of 40 for a reasonably comparable visualization (128^2 particles, 10^2 point sprite size). In general, forward particle-based integration is extremely advantageous for long particle lifetimes, while it loses some of its efficiency for a very dense set of particles with significant overdraw.

A time-dependent visualization needs an additional transfer of flow data from main memory to the GPU for each time step. The performance overhead only depends on data size. For example, for 128^2 particles with 10^2 point sprite size, 16-bit floating-point flow data sets, and a fixed viewport size of 640×480, the overall performance decreases from 444.1 fps for steady flow to 44.5 fps or 161.4 fps for unsteady flow with a flow resolution of 640×480 or 320×240, respectively.

Table 2 documents the performance for varying size of the sprite texture, when the size of the point sprite is fixed in image space. The numbers indicate that textures of size 32^2 or less fit into texture cache because no performance increase is achieved by reducing the texture size below 32^2. Therefore, we use point sprite textures of size 32^2 for all applications as an optimal compromise between texture resolution and speed.

Table 3 shows performance measurements for particle-in-cell advection. The size of the property texture is identical to the number of particles. The numbers demonstrate that the computation time is approximately linear with the number of particles and that alpha blending between successive time steps has a small effect when a reasonably large number of particles is employed.

Table 4 documents the performance of the convolution stage of the framework. Different filter lengths and viewport sizes are compared, showing an approximately linear behavior with respect to the number of texels and the number of integration steps. For example, this convolution has to be explicitly performed for the DLIC and UFAC realizations within the framework. Here, the overall computation time is determined by adding the computational costs for convolution and particle advection (see Table 1).

Table 4: Performance for convolution (in fps).

Viewport	Filter Length		
	20	40	80
640×480	61.1	33.2	17.2
1600×1200	10.8	5.7	2.8

Table 1: Performance of the particle system for integration, rendering, and blending (in fps).

# Particles	w/o VP	Point Sprite Size			
		0^2	5^2	10^2	20^2
64^2	1968.3	1380.4	1285.9	1012.8	565.1
128^2	1450.3	831.0	712.9	444.1	186.1
256^2	681.4	323.0	253.1	137.9	51.2
512^2	191.1	84.7	68.0	35.5	12.8
1024^2	47.3	22.3	16.7	8.9	3.2

Table 2: Performance for a point sprite size of 10^2 pixels (in fps).

Texture Size	# Particles	
	64^2	256^2
8^2	1015.1	137.9
16^2	1014.2	137.9
32^2	1012.8	137.9
64^2	663.6	66.0
128^2	424.4	34.5

Table 3: Performance for particle-in-cell advection (in fps).

# Particles	Alpha Blending	
	Disabled	Enabled
64^2	1095.9	796.6
128^2	753.2	600.0
256^2	323.0	291.5
512^2	90.6	87.6
1024^2	22.9	22.7

11 Conclusion and Future Work

We have presented a new hybrid particle and texture based approach for the visualization of time-dependent vector fields. A particle-based representation overcomes efficiency and memory problems of our previous implementation of a spacetime framework for time-dependent vector field visualization because trajectories can now be constructed incrementally and a simultaneous access to many different time steps is avoided. We have described a probabilistic particle injection and removal mechanism to maintain a given particle density. As an alternative, particle-in-cell advection provides a deterministic density control, with exactly one particle per cell. We have presented an efficient GPU implementation of our approach that facilitates the interactive visualization of unsteady 2D flow on Shader Model 3 compliant graphics hardware. Another advantage of the hybrid particle/texture framework is high visualization quality, achieved by accurate Lagrangian integration.

Our framework achieves spacetime-coherent dense representations by a two-step process: construction of continuous trajectories in spacetime for temporal coherence, and convolution along another set of paths through the above spacetime volume for spatially correlated patterns. We have demonstrated the flexibility of the framework by mimicking LEA, DLIC, and UFAC and we have explicitly stated what visual structures are constructed by these different approaches. In this context, generic strategies for dense unsteady flow visualization have been analyzed.

For future work, an extension to 3D vector fields will be most challenging. Particle representation, Lagrangian integration, and density control are already formulated in a dimension-independent manner and could be easily used for 3D flow. More difficult, however, will be the development of methods for fast 3D convolution and for an efficient scan conversion of RBFs on a 3D uniform grid. In addition, perceptual issues (occlusion, clutter, spatial perception) have to be addressed for dense 3D representations. Nevertheless, we are convinced that our approach is immediately interesting for 3D flow visualization on curved surfaces: Recent methods for curved manifolds [11, 24, 28] are based on 2D image space, where our 2D framework implementation could be adopted.

Acknowledgments

The data of the Gulf of Mexico was made available courtesy of Dr. O'Brien (COAPS/FSU), and the hurricane Dennis data was kindly provided by Dr. X. Zou (Meteorology/FSU). G. Erlebacher acknowledges support from the Program Enhancement Grant program at FSU. Thanks to Simon Stegmaier for fruitful discussions on texture advection.

References

[1] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proc. ACM SIGGRAPH 93*, pages 263–270, 1993.

[2] L. Devroye. *Non-Uniform Random Variate Generation*. Springer, Berlin, 1986.

[3] G. Erlebacher, B. Jobard, and D. Weiskopf. Flow textures: High-resolution flow visualization. In C. D. Hansen and C. R. Johnson, editors, *The Visualization Handbook*, pages 279–293. Elsevier, Amsterdam, 2005.

[4] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *Proc. IEEE Vis.*, pages 155–162, 2000.

[5] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Trans. Vis. Comput. Gr.*, 8(3):211–222, 2002.

[6] B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. *Comput. Gr. Forum (Eurographics 2000)*, 19(3):31–40, 2000.

[7] P. Kipfer, M. Segal, and R. Westermann. Uberflow: A GPU-based particle engine. In *Proc. Gr. Hardw.*, pages 115–122, 2004.

[8] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proc. Gr. Hardw.*, pages 123–132, 2004.

[9] J. Krüger, P. Kipfer, P. Kondratieva, and R. Westermann. A particle system for interactive visualization of 3D flows. *IEEE Trans. Vis. Comput. Gr.* Accepted for publication.

[10] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Comput. Gr. Forum*, 23(2):143–161, 2004.

[11] R. S. Laramee, B. Jobard, and H. Hauser. Image space based visualization of unsteady flow on surfaces. In *Proc. IEEE Vis.*, pages 131–138, 2003.

[12] W. Lefer, B. Jobard, and C. Leduc. High-quality animation of 2D steady vector fields. *IEEE Trans. Vis. Comput. Gr.*, 10(1):2–14, 2004.

[13] G.-S. Li, U. Bordoloi, and H. W. Shen. Chameleon: An interactive texture-based framework for visualizing three-dimensional vector fields. In *Proc. IEEE Vis.*, pages 241–248, 2003.

[14] Z. Liu and R. J. Moorhead II. Accelerated unstead flow line integral convolution. *IEEE Trans. Vis. Comput. Gr.*, 11(2):113–125, 2005.

[15] N. Max and B. Becker. Flow visualization using moving textures. In *Proc. ICASW/LaRC Symp. Vis. Time-Varying Data*, pages 77–87, 1995.

[16] A. Okada and D. Lane. Enhanced line integral convolution with flow feature detection. In *Proc. SPIE Vol. 3017 Visual Data Exploration and Analysis IV*, pages 206–217, 1997.

[17] F. P. Pighin, J. M. Cohen, and M. Shah. Modeling and editing flows using advected radial basis functions. In *EG/SIGGRAPH Symp. Comput. Anim.*, pages 223–232, 2004.

[18] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Comput. Gr. Forum*, 14(4):181–188, 1995.

[19] H.-W. Shen and D. L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Trans. Vis. Comput. Gr.*, 4(2):98–108, 1998.

[20] A. Sundquist. Dynamic line integral convolution for visualizing streamline evolution. *IEEE Trans. Vis. Comput. Gr.*, 9(3):273–283, 2003.

[21] A. Telea and J. J. van Wijk. 3D IBFV: Hardware-accelerated 3D flow visualization. In *Proc. IEEE Vis.*, pages 233–240, 2003.

[22] J. J. van Wijk. Spot noise – texture synthesis for data visualization. *Comput. Gr. (Proc. ACM SIGGRAPH 91)*, 25:309–318, 1991.

[23] J. J. van Wijk. Image based flow visualization. *ACM Trans. Gr.*, 21(3):745–754, 2002.

[24] J. J. van Wijk. Image based flow visualization for curved surfaces. In *Proc. IEEE Vis.*, pages 123–130, 2003.

[25] R. Wegenkittl, E. Gröller, and W. Purgathofer. Animating flow fields: Rendering of oriented line integral convolution. In *Comput. Anim. '97*, pages 15–21, 1997.

[26] D. Weiskopf. Texture-based flow visualization. `http://www.vis.uni-stuttgart.de/texflowvis`, 2005.

[27] D. Weiskopf, G. Erlebacher, and T. Ertl. A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *Proc. IEEE Vis.*, pages 107–114, 2003.

[28] D. Weiskopf and T. Ertl. A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *Proc. Gr. Interface*, pages 263–270, 2004.

[29] 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 *Proc. VMV '01*, pages 439–446, 2001.

[30] D. Weiskopf, T. Schafhitzel, and T. Ertl. Real-time advection and volumetric illumination for the visualization of 3D unsteady flow. In *Proc. Eurovis (EG/IEEE TCVG Symp. Vis.)*, pages 13–20, 2005.

[31] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumes. In *Proc. IEEE Vis.*, pages 417–424, 2003.