A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields

Daniel Weiskopf[1] Gordon Erlebacher[2] Thomas Ertl[1]

[1]Institute of Visualization and Interactive Systems, University of Stuttgart[*]

[2]School of Computational Science and Information Technology, Florida State University[†]

Abstract

We propose Unsteady Flow Advection–Convolution (UFAC) as a novel visualization approach for unsteady flows. It performs time evolution governed by pathlines, but builds spatial correlation according to instantaneous streamlines whose spatial extent is controlled by the flow unsteadiness. UFAC is derived from a generic framework that provides spacetime-coherent dense representations of time dependent-vector fields by a two-step process: 1) construction of continuous trajectories in spacetime for temporal coherence, and 2) convolution along another set of paths through the above spacetime for spatially correlated patterns. Within the framework, known visualization techniques—such as Lagrangian-Eulerian Advection, Image-Based Flow Visualization, Unsteady Flow LIC, and Dynamic LIC—can be reproduced, often with better image quality, higher performance, or increased flexibility of the visualization style. Finally, we present a texture-based discretization of the framework and its interactive implementation on graphics hardware, which allows the user to gradually balance visualization speed against quality.

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: time-dependent vector fields, unsteady flow visualization, LIC, texture advection, hardware acceleration

1 Introduction

An important class of flow visualization techniques computes the motion of massless particles transported along the velocity field to obtain characteristic structures like streamlines, pathlines, or streaklines. A fundamental issue is an appropriate placing of seed points for particle tracing. Critical vector field features will be missed if these important regions are not covered by any particle path. This problem can be overcome by a dense representation, i.e., by covering the domain densely with particle traces.

In this paper, we adopt the concept of a dense representation and specifically address the visualization of time-dependent vector fields. A crucial observation is that two types of coherence are essential for understanding animated visualizations. First, spatial coherence within a single picture reveals image patterns indicating some vector field structure. Second, frame-to-frame coherence allows the user to identify a consistent motion of these patterns. Many previous approaches for time-dependent vector field visualization do not explicitly model spatial and temporal coherence, but intermingle these two distinct features. To achieve separate control over both properties we apply a generic and flexible framework that provides spacetime-coherent dense representations of time-dependent vector fields by a two-step process. First, continuous trajectories are constructed in spacetime to guarantee temporal coherence. Second, convolution along another set of paths through the above spacetime results in spatially correlated patterns. The framework can be mapped to a discrete texture-based representation in combination with a SIMD architecture and therefore makes feasible an efficient implementation on graphics hardware.

The actual visualization result depends on the choice of parameters, leading to a wide range of possible applications. For example, the framework is capable of mimicking known visualization techniques, such as Line Integral Convolution (LIC) for steady vector fields [Cabral and Leedom 1993], Unsteady Flow LIC (UFLIC) [Shen and Kao 1998], Lagrangian-Eulerian Advection (LEA) [Jobard et al. 2002], Image-Based Flow Visualization (IBFV) [van Wijk 2002], and Dynamic LIC (DLIC) [Sundquist 2003]. Although the same qualitative visualization results are achieved, the underlying computational models may differ significantly, causing differences with respect to image quality, performance, and flexibility. Since the framework makes use of two different sets of paths for temporal and spatial coherence, even more advanced visualization techniques have become possible. We propose Unsteady Flow Advection–Convolution (UFAC) as a novel approach for unsteady fluid flows that performs time evolution governed by pathlines, but builds spatial correlation according to instantaneous streamlines. The spatial extent of streamlines is related to the degree of unsteadiness.

2 Previous Work

The discussion of previous work focuses on noise-based and dense texture representations [Sanna et al. 2000]. An early texture-synthesis technique for vector field visualization, spot noise [van Wijk 1991], generates a texture by distributing a large number of spots on the spatial domain. LIC [Cabral and Leedom 1993] is a popular technique for the dense representation of streamlines in steady vector fields. LIC locally smoothes an input noise texture along streamlines by convolution with a filter kernel, leading to a high correlation along, and little or no correlation perpendicular to, streamlines. Subsequently, LIC has been extended in various respects: improvement of contrast and quality by postprocessing [Okada and Kao 1997], animated LIC [Forssell and Cohen 1995], visualization of the orientation of flow [Wegenkittl et al. 1997], the combination of animation and dye advection [Shen et al. 1996], and Fast LIC [Stalling and Hege 1995]. Pseudo LIC (PLIC) [Verma et al. 1999] uses LIC to generate a template texture that is then

[*]{weiskopf,ertl}@informatik.uni-stuttgart.de

[†]erlebach@csit.fsu.edu

mapped onto "thick" streamlines. UFLIC [Shen and Kao 1998] and its accelerated version [Liu and Moorhead 2002] address the issue of temporal coherence by scattering particles along pathlines and subsequently collecting their contributions to the filtered image. DLIC [Sundquist 2003] is an extension of LIC that allows for the evolution of streamlines in time-dependent vectors fields, such as electric fields. Streamline evolution is also considered in the context of evenly spaced streamlines [Jobard and Lefer 2000].

Another, yet related class of dense representations is based on texture advection. The basic idea is to represent a dense collection of particles in a texture and transport this texture according to the motion of particles [Max and Becker 1995]. LEA [Jobard et al. 2002] visualizes unsteady flows by integrating particle positions (i.e., the Lagrangian part) and advecting the color of the particles based on a texture representation (i.e., the Eulerian aspect). IBFV [van Wijk 2002] is a recently developed variant of 2D texture advection. Not only is the texture transported along the flow, but additionally a second texture is blended into the advected texture at each time step. In an alternative approach, nonlinear diffusion can be used to visualize particle transport along a flow field [Diewald et al. 2000].

Implementations on graphics hardware are possible for many of the above techniques to increase performance. For example, GPU (graphics processing unit) based implementations are known for LIC [Heidrich et al. 1999], for different versions of texture advection [Jobard et al. 2000; Weiskopf et al. 2001; Weiskopf et al. 2002], and for IBFV [van Wijk 2002].

3 Continuous Framework

We briefly discuss a generic framework that describes spatial and temporal correlation for vector field visualization on a continuous level. Most parts of the framework are explained in more detail in [Erlebacher et al. 2003]. The development of the framework was inspired by a similar approach taken for DLIC [Sundquist 2003].

In what follows, a 2D spatial domain is assumed because the visualization applications of this paper are restricted to 2D. However, the framework itself can be directly extended to 3D. In a flat, Euclidean 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. Pathlines $\mathbf{x}_{\text{path}}$ are the integral curves of a vector field, governed by the ordinary differential equation

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

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 two spatial dimensions. We assume changes of reference frames to be governed by the Galilei group of Newtonian physics. The world line (i.e., 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 three components: two 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. 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 trajectories.

Properties can be attached to particles to distinguish them from one another. In what follows, a property is assumed to be a grayscale value from the range $[0,1]$. Properties are allowed to change continuously along the trajectory; very often, however, they remain constant. From a collection of trajectories, along with the corresponding particle properties, a function $I(\mathbf{x},t)$ can be defined by setting its value to the property of the particle crossing through the spacetime point $(\mathbf{x},t)$. The function value will be zero if no trajectory touches the point. The function $I(\mathbf{x},t)$ filling spacetime is an important element of the framework. The continuous behavior of trajectories and their attached properties ensures that spatial slices through the property field $I(\mathbf{x},t)$ at nearby times are strongly related—and therefore temporal coherence is achieved. An animated sequence built from spatial slices with uniformly increasing time yields 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

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

along a path $\mathscr{Z}(s;\mathbf{x},t)$ through spacetime. The subscript on D_t is a reminder that the filtered image depends on time. For generality, we perform the convolution along the entire time axis, and rely on the definition of the filter kernel $k(s)$ to restrict the domain of integration. The kernel need not be the same for all points on the filtered slice and may depend on additional parameters, such as data derived from a vector field. For simplicity of notation, these parameters are not explicitly marked in the mathematical expressions. $\mathscr{Z}(s;\mathbf{x},t)$ 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 another vector field $\mathbf{w}(\mathbf{x},t)$. The temporal component of $\mathscr{Z}(s;\mathbf{x},t)$ may depend on s and t. In the visualization approaches of this paper, either $\mathscr{Z}(s;\mathbf{x},t) = (\,\cdot\,,t)$ or $\mathscr{Z}(s;\mathbf{x},t) = (\,\cdot\,,s)$ are used.

An animated visualization produces filtered images D_t for uniformly increasing times t. Two issues have to be considered for a useful animation sequence. First, each single filtered image should introduce some intuitively understandable visual patterns. Therefore, the property field I and the family of convolution paths $\mathscr{Z}$ should not be chosen arbitrarily, but should be constructed to lead to these patterns. Secondly, the spatial structures should be coherent from frame to frame. It is important to note that the coherence of I along the time axis does not automatically imply a temporal coherence of the filtered images D_t. Thus, further restrictions are imposed on the choice of I and $\mathscr{Z}$.

In summary, the structure of D_t is defined by the triplet $[I,\mathscr{Z},k]$, where I is built from trajectories $\mathscr{Y}$; some useful combinations for this triplet are discussed in Section 5. The main advantage of this generic framework is its direct, explicit, and 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 $\mathbf{w}(\mathbf{x},t)$.

4 Texture-Based Discretization

The continuous framework must be discretized to compute images. We chose a novel texture-based approach for storing and processing the dense representation, the vector field itself, and some additional auxiliary data. Since the algorithms only utilize textures and no other complex data structures, an efficient, purely GPU-based implementation is possible. The rough outline of the texture-based discretization is as follows. First, spatial slices of the property field I are constructed from trajectories. For each texel, trajectories backward in time are iteratively computed in a Lagrangian manner; property contributions from particles starting at previous times and propagating to the current texel position of I are collected along the trajectory. By combining spatial slices, the complete property field defined on a spacetime domain is built. Second, a convolution is computed along $\mathscr{X}$ within the resulting 3D property field. A Lagrangian integration of the corresponding paths is iteratively calculated to obtain contributions for the respective texel.

A more detailed description of the discretization follows, starting with the first major component—the construction of the property field. Still in a continuous notation, we assign to each particle a finite lifetime τ, and therefore each trajectory $\mathscr{X}$ has a maximum temporal length. A particle is born at $(\mathbf{x}_0, t_0)$ and dies at time $t = t_0 + \tau$. We describe newly introduced particles by a field $P(\mathbf{x}, t)$ defined on the spacetime domain. The field yields a continuous description—particles are no longer considered as point objects, but rather are continuously spread over space, i.e., the particle field describes a particle density. The invariant property of a particle born at $(\mathbf{x}_0, t_0)$ is described by $P(\mathbf{x}_0, t_0)$. The actual property of a particle may change during its lifetime, governed by a weight function $w(t_{\text{age}})$, where $t_{\text{age}} \in [0, \tau]$ is the age of the particle. For example, $w = 1$ describes a constant property along the entire trajectory, while some other choice for the weight may be used for a gradual phase-in and phase-out of a particle's brightness. The weight might also depend on outside parameters, such as information derived from the data set or the particle injection field P. These parameters are not explicitly noted. The contribution to the property field I at time t by a particle born at position $\mathbf{x}_0$ and at an earlier time t_0 (within the lifetime) is therefore

$$I(\mathscr{Y}(t; \mathbf{x}_0, t_0)) = w(t - t_0) P(\mathbf{x}_0, t_0) \quad .$$

Due to the symmetry of Newtonian mechanics with respect to time reversal, the trajectory can equally well be computed backward in time, starting from t. By adding contributions from all possible particle paths originating from different discrete times t_l between $(t - \tau)$ and t, one obtains the property field

$$I(\mathbf{x}, t) = \sum_{t_l} w(t - t_l) P(\mathscr{Y}(t_l; \mathbf{x}, t)) \quad .$$

The fields I and P are discretized by a uniform sampling in space and time. Stated differently, I and P can be represented by stacks of equidistant 2D textures. We regard textures as uniform grids, and texels as cells of these grids. Accordingly, the discrete property field is

$$I_i(\mathbf{x}_{jk}) = \sum_{l=i}^{i-\Delta i_\tau} w(t_i - t_l) P_l(\mathbf{x}_{\text{path}}(t_l; \mathbf{x}_{jk}, t_i)) \quad .$$

Here, I_i and P_i denote spatial slices of I or P at time t_i, respectively; Δi_τ is the maximum number of discrete time intervals per lifespan; $\mathbf{x}_{jk}$ denotes the spatial position of a grid point. Being a texture, P_i can represent any distribution of released particles—ranging from white noise to frequency-filtered noise or some input with large-scale structures that is used for dye injection. Figure 1 illustrates how particles released at previous times contribute to the property at time t_i. The property slice I_i is only sampled at discrete grid points $\mathbf{x}_{jk}$, i.e., point sampling is assumed. Without changing the basic procedure, more complex filtering, for example by means of subsampling, could be applied. Due to convergence or divergence of the vector field, trajectories starting at a given spatial distance from each other might sample the injection textures P_i at varying distances (i.e., at different spatial frequencies). However, the limited lifetime of particles ensures that these differences do not grow without bound (provided that the vector field is smooth) and therefore subsampling and filtering is usually not required.

Although the access to I_i is restricted to grid points $\mathbf{x}_{jk}$, the lookup in P_l is not. Therefore, some kind of interpolation scheme has to be employed. In our implementation, a tensor product of

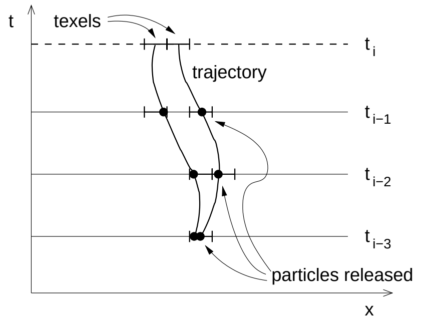

Figure 1: Computing two example texels of I_i (dashed horizontal slice) by adding all possible contributions from particles starting (marked by black dots) at previous times.

linear interpolations is used, i.e., a bilinear interpolation on 2D spatial slices. Alternatively, simple nearest-neighbor sampling or more sophisticated higher-order interpolation schemes are possible. In all discussions of this paper, the existence of some interpolation scheme is assumed to access textures at arbitrary positions.

Finally, the discretized pathline $\mathbf{x}_{\text{path}}(t_l; \mathbf{x}_{jk}, t_i)$ is computed from the vector field, which is stored either as a stack of 2D textures for different times or as a 3D texture for a spacetime domain. Furthermore, we assume that some black-box function

$$\Delta\mathbf{x}(\mathbf{x}, t_i) = \mathbf{x} - \mathbf{x}_{\text{path}}(t_i - \Delta t; \mathbf{x}, t_i) \tag{3}$$

yields the motion of a particle for a small time step Δt. In our current implementation, first-order explicit Euler integration approximates $\Delta\mathbf{x}(\mathbf{x}, t_i)$ by $\Delta t\mathbf{u}(\mathbf{x}, t_i)$. Higher-order schemes could be applied if higher accuracy were needed.

Combining the aforementioned building blocks, the texture-based computation of a single spatial slice I_i can be accomplished by iterating Δi_τ times the recurrence relations

$$\begin{aligned} \tilde{\mathbf{X}}_l(\mathbf{x}_{jk}) &= \tilde{\mathbf{X}}_{l+1}(\mathbf{x}_{jk}) - \Delta\mathbf{x}(\tilde{\mathbf{X}}_{l+1}(\mathbf{x}_{jk}), t_{l+1}) & (4) \\ \tilde{I}_l(\mathbf{x}_{jk}) &= \tilde{I}_{l+1}(\mathbf{x}_{jk}) + w(t_i - t_l) P_l(\tilde{\mathbf{X}}_l(\mathbf{x}_{jk})) \quad , & (5) \end{aligned}$$

with $l = i-1, \ldots, i - \Delta i_\tau$. The initial values are

$$\tilde{\mathbf{X}}_i(\mathbf{x}_{jk}) = \mathbf{x}_{jk} \quad \text{and} \quad \tilde{I}_i(\mathbf{x}_{jk}) = w(0) P_i(\mathbf{x}_{jk}) \quad . \tag{6}$$

The final result is

$$I_i(\mathbf{x}_{jk}) = \tilde{I}_{l-\Delta i_\tau}(\mathbf{x}_{jk}) \quad .$$

The 2D texture $\tilde{I}_l$ holds the intermediate results for the accumulation of property values; $\tilde{\mathbf{X}}_l$ is a two-component 2D texture that stores the iteratively updated spatial positions along pathlines. Figure 1 illustrates this process in a spacetime diagram. The iteration starts at the current time t_i and scans starting points of trajectories at previous time steps, up to the maximum lifetime (which is three time steps in this example). The lookups in $\tilde{\mathbf{X}}_l$ and $\tilde{I}_l$ are restricted to fixed grid points $\mathbf{x}_{jk}$ and do not need an interpolation scheme. Since the above iteration equations and $\Delta\mathbf{x}(\mathbf{x}, t)$ only need access to various textures and a fixed set of numerical operations, a direct mapping to fragment processing on modern graphics hardware is feasible. Specific GPU-related implementation issues are discussed in Section 6.

So far, a uniform lifetime τ was assumed. However, the lifetime of particles can be individually adapted by changing the support of the weight function $w(t_{\text{age}})$—as long as some maximum age $\tau_{\max}$ is not exceeded. Since a large and possibly varying number of particles contributes to each texel of I_i, the sum of the weights w should be normalized to one for each cell of I_i. Therefore, the weights are accumulated along with the accumulated properties $\tilde{I}_l$, and the final property value I_i is the accumulated property divided by the accumulated weight.

One issue concerns inflow regions, i.e., parts of the domain in which the backward integration of trajectories leaves the boundaries of the textures P_i. The solution is to enlarge the input textures so that the pathlines do not exceed these boundaries. The texture size depends both on the maximum magnitude of the vector field and on the lifetime of the particles. Typically, the particle injection textures P are enlarged only virtually by applying texture wrapping, provided that these textures are seamless, e.g., white noise. Similarly, the vector field textures can be virtually enlarged by clamping texture coordinates to the texture boundaries. Another issue is a possible change of spatial frequency of input P along trajectories through regions of divergence or convergence. However, due to the limited lengths of trajectories, these effects do not continue over large distances and are usually not noticeable even in small regions. Finally, the accumulation of random particles released from various previous times in a noise-based approach (i.e., uncorrelated injection with respect to both space and time) essentially represents a summation of independent random variables, leading to a normal distribution dictated by the central limit theorem. The standard deviation depends on the number of elements added, i.e., on Δi_τ. For a long lifetime of particles, I_i becomes a gray image with most intensities in a small interval around 0.5. Good contrast can be reestablished by histogram equalization [Okada and Kao 1997]. It is sufficient to compute the histogram once because the distribution of property values changes only slightly over time.

The second major component of the framework concerns the convolution along spacetime paths $\mathscr{Z}$. The basic idea is to discretize the convolution integral (2) by a Riemann sum. Fortunately, the texture-based implementation of this part is strongly related to the above implementation of the construction of I. In what follows, analogous elements in the two implementations are briefly described and differences are discussed. Since the spatial components of both $\mathscr{Y}$ and $\mathscr{Z}$ are based on pathlines of some vector field, the integration process acting on coordinate textures $\tilde{\mathbf{X}}_i$ is the same. The filter kernel $k(s)$ replaces the weight $w(t_{\text{age}})$, including the accumulation of kernel contributions and the final normalization to unity. Moreover, the filter kernel can depend on the position in spacetime, additional outside parameters, or values derived from the given data set. Input to the convolution process is the stack of properties I_i, which acts as a substitute for the description of particle injections, P_i. The maximum lifetime of particles is analogous to the maximum support of the filter kernel.

The following differences exist between constructing I and computing the convolution. First, the vector fields for computing pathlines can be different. Second, the temporal component of $\mathscr{Z}(s;\mathbf{x},t)$ is not necessarily s. Therefore, a mechanism has to calculate the appropriate time t_i for each iteration step and address the corresponding slice I_i. In all visualization approaches investigated so far, the temporal component of $\mathscr{Z}(s;\mathbf{x},t)$ is either s or t, i.e., the iteration either processes the stack of textures I_i concurrently to the integration time or uses a single property texture at a fixed time. Third, a provision must be made for integrating the convolution forward and backward in time, which is implemented by performing both parts independently and combining their results. Fourth, the treatment of boundaries is different. The convolution integral is limited to the boundaries of the domain; once the integrated coordinates leave the domain, the kernel is set to zero. Due to the normalization of accumulated kernel weights, the property values exhibit no intensity loss by a decreased filter support near boundaries. Fifth, both histogram equalization, which is also used for reestablishing high contrast in the construction of I, and edge enhancement by high-pass filtering (such as a Laplacian filter) are used to improve image quality [Okada and Kao 1997].

The main features of this implementation of the continuous framework are the following. First, an explicit and separate control over the temporal evolution and the structure of spatial patterns is possible. Second, any input texture P_i can be used; one is not restricted to white noise or to dye textures that exhibit only large-scale spatial structures. Third, a complete Lagrangian integration scheme is employed for particle transport in each time step of an animation, which avoids the accumulation of sampling artifacts or artificial smoothing, prevalent in Eulerian or semi-Eulerian approaches. Fourth, there is no restriction on the possible filter kernel $k(s)$ or property weight $w(t_{\text{age}})$—except for a compact support of these functions. They can even depend on parameters derived from data sets or current spacetime locations. This flexibility allows for custom kernel design for further quality enhancement. A similar argument holds for the temporal behavior of particle transport along $\mathscr{Y}$; for example, popping artifacts can be avoided by a slow phase-in and phase-out of particles through a time-dependent property weight.

5 Visualization Approaches

Within the structure of the above framework and its texture-based discretization, several visualization approaches are possible. In this section, some well-known visualization techniques for time-dependent data sets are revisited. Detailed background information on the mapping of LEA, IBFV, and DLIC to the continuous framework is presented in [Erlebacher et al. 2003]. The basic idea is that by choosing appropriately the parameters at our disposal, visualization results that are qualitatively similar to existing techniques can be reproduced, even if the underlying computational models differs significantly. We focus in particular on differences with respect to image quality, performance, and flexibility. Furthermore, we propose a new approach: the Unsteady Flow Advection–Convolution derived from our framework.

5.1 Lagrangian-Eulerian Advection

LEA [Jobard et al. 2002] is realized within the framework by setting $\mathbf{u}(\mathbf{x},t)$ to the given vector field $\mathbf{v}(\mathbf{x},t)$, and $\mathbf{w}(\mathbf{x},t) = 0$. LEA does not change the properties of particles along their paths (i.e., $w(t_{\text{age}}) = 1$). Particle transport is computed according to the description in Section 4. To emulate the exponential filter of the original version of LEA, successive α blending of the results of particle transport can be applied. For any other kernel, the implementation of Section 4 is employed—except for the superfluous update of positions $\tilde{\mathbf{X}}_l(\mathbf{x}_{jk})$. The advantage of the exponential filter is that only one blending operation is required for each final image D_i, while other kernels require an iterative computation of the entire convolution for each time step. However, some of these successive filter calculations could be accelerated by exploiting coherence analogously to FLIC methods [Stalling and Hege 1995].

Particle transport in the original LEA utilizes a hybrid Lagrangian-Eulerian texture update. Unfortunately, this computation scheme does not always reproduce trajectories faithfully and white noise input becomes gradually duplicated over several texels. Therefore, larger homogeneous patterns, even perpendicular to pathlines, emerge after some time of particle motion. The degradation of noise frequency is overcome by an ad-hoc injection of noise during each time step. Furthermore, many particles are completely discarded from one time step to the following because a texel is only filled with information from a single texel of the previous time step (by nearest-neighbor sampling). As a consequence, the original LEA exhibits only suboptimal temporal correlation, which is apparent in the form of noisy and rather short spatial patterns after convolution. For the implementation of LEA within our framework, these issues do not arise and therefore the final visualization shows an improved temporal and spatial coherence. As another advantage, the framework supports any type of input texture P_i, and is not restricted to white noise as in the original LEA. Thus, high spatial frequencies can be removed in a prefiltered input to avoid aliasing artifacts after convolution.

The disadvantage of the alternative implementation is its higher computational cost for processing Δi_τ input images per time step. However, for short lifetimes of particles (e.g., some 15 iterations), alternative LEA executes at interactive frame rates on current GPUs and yet shows a higher temporal correlation than the original implementation. Moreover, the framework can be easily adjusted to targeted frame rates by adapting Δi_τ; for example, only a few iteration steps could be processed for particle transport during user interaction, while a high-quality update of a snapshot could be computed with an increased number of iterations.

5.2 Image-Based Flow Visualization

In the context of the framework, IBFV [van Wijk 2002] is dual to LEA: now $\mathbf{w}(\mathbf{x},t)$ is identified with the given vector field $\mathbf{v}(\mathbf{x},t)$, and $\mathbf{u}(\mathbf{x},t) = 0$. From a physical point of view, IBFV shows streaklines of particles injected at various points according to the structure of the input P_i.

The implementation of IBFV within the framework computes the convolution according to the description in Section 4. The construction of the property field I is trivial: I_i is set to the particle injection at the same time, P_i. The only essential difference between the original IBFV implementation and the alternative framework-based implementation is the convolution process. The original approach permanently and implicitly updates $\tilde{I}_l$, i.e., particles are resampled on a texture after each integration step. This resampling is based on bilinear interpolation and thus results in artificial smoothing. In contrast, the alternative IBFV implementation never resamples particles for intermediate steps, but rather employs a Lagrangian integration of the complete pathline and samples only the original particles released on P_i. Therefore, no artificial diffusion is introduced.

The alternative implementation has the downside of higher computational costs because complete pathlines have to be calculated for each time step. We think that a combination of both implementations is most useful: interactive visualization could be based on the faster original implementation, while high-quality, high-contrast snapshots with very long streaklines are produced by the alternative approach. At no additional cost, filters different from an exponential kernel (which is intriniscally built into the original IBFV) can be applied for these snapshots to further improve image quality or achieve different visualization styles.

5.3 Unsteady Flow LIC

UFLIC [Shen and Kao 1998] visualizes unsteady flow fields by advecting a dense set of particles forward in time—particles are released in a noise-based fashion and then distributed over space and time. The original UFLIC implementation attaches a time stamp to each particle to feed the points of a trajectory into spacetime buckets. Figure 2 illustrates for a single trajectory how points are distributed into buckets of temporal size ΔT. Using forward integration and scattering, several buckets next to each other in a single spacetime slab (a subset of spacetime covering the entire spatial domain and having a temporal thickness of ΔT) can be filled by spacetime curve drawing. At each time step T_j (i.e., an integer multiple of ΔT), a large number of new particles is released and traced up to the maximum age τ to obtain a dense representation. The final image for one spacetime slab is constructed from the contributions

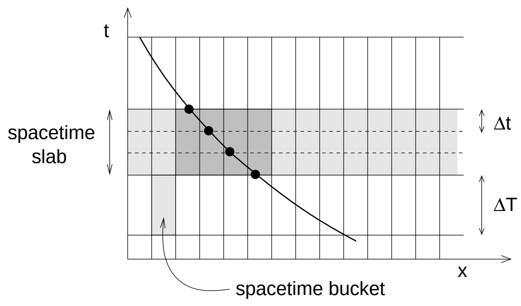

Figure 2: Spacetime sampling of trajectories for UFLIC.

of all particles crossing this slab. Actual visualization results are displayed at time intervals ΔT.

Unlike the original UFLIC implementation, our framework is completely texture-based and cannot make use of scattering or curve-based modeling. However, the same qualitative visualization approach is feasible by subsampling spacetime slabs at intervals Δt that are shorter than ΔT, see Figure 2. In the formalism of the framework, $\mathbf{u}(\mathbf{x},t)$ is set to the given vector field $\mathbf{v}(\mathbf{x},t)$, and $\mathbf{w}(\mathbf{x},t) = 0$. Convolution is performed along straight lines parallel to the time axis, and the support of the filter is exactly the thickness of a spacetime slab, ΔT. Particle injection textures P_j are only needed at the coarser steps T_j. The original UFLIC approach reuses filtered images as input for releasing new particles to achieve highly correlated spatial structures. This feed-forward technique can be directly mapped to the framework by using D_i as a particle injection texture P_l for a later time—at the expense of losing a clear separation between the mechanisms of spatial and temporal correlations.

5.4 Dynamic LIC

DLIC [Sundquist 2003] targets the animation of instantaneous streamlines (or fieldlines) of a time-dependent vector field, e.g., an electric field. More specifically, two vector fields are given: $\mathbf{w}(\mathbf{x},t)$ governs the streamline generation at each time step, while $\mathbf{u}(\mathbf{x},t)$ describes the evolution of streamlines. Therefore, $\mathbf{w}(\mathbf{x},t)$ and $\mathbf{u}(\mathbf{x},t)$ cannot be chosen independently. Using these variable names for the vector fields, DLIC maps one-to-one to the framework. Unlike the aforementioned techniques, DLIC sets the temporal component of the convolution lines to the time corresponding to the filtered image, i.e., $\mathscr{X}(s;\mathbf{x},t') = (\,\cdot\,,t')$. In this way, convolution is computed on a single spatial slice through I.

In the original CPU-based implementation of DLIC, a large number of individual particles, representing discs of finite radius, is tracked in a Lagrangian approach. The particles are allowed to overlap. An almost uniform particle density is achieved by removing or adding particles on a per-cell basis after each integration step. A noise texture is produced by sampling the particles on a uniform grid. Finally, streamlines are generated by FLIC [Stalling and Hege 1995]. The main advantage of the implementation within the framework is GPU support and thus an interactive visualization speed. (Sundquist [2003] does not report performance measurements, but we assume a speed well below interactive frame rates for typical data set sizes.) Another advantage is related to the LIC process. Sundquist takes special care in choosing seed points for FLIC to avoid temporal sampling artifacts. Unlike FLIC, the convolution in the framework employs an independent sampling at every texel and integration step, and therefore is not subject to these artifacts. Both implementations may exhibit problems in regions of strong convergence or divergence. The original DLIC algorithm loses contrast for contracting fields, while the spatial frequency of noise might be changed in our implementation (see Section 4).

5.5 Unsteady Flow Advection–Convolution

Unsteady Flow Advection–Convolution (UFAC) is a novel approach for displaying unsteady fluid flows. The development of this method has been guided by the following observations. First, streaklines of an unsteady flow (or similarly, pathlines) may cross each other and, consequently, are an inappropriate basis for a dense representation. Second, streamlines never intersect (except at critical points). Third, a coherent temporal evolution of image patterns can be recognized by the user as a consistent motion, and can be used as an additional means of conveying paths. Fortunately, the spatial part of these paths may intersect at different times without destroying the impression of a consistent motion.

UFAC aims to produce an animated sequence of streamline images, where the evolution of streamlines is governed by the motion of particles along pathlines. In this way, each image would contain only non-intersecting line structures, while the temporal evolution would reveal the trajectories of particles. Analogously to DLIC, this approach can be formulated in the context of the framework by setting $\mathbf{u}(\mathbf{x},t)$ to the given time-dependent flow field $\mathbf{v}(\mathbf{x},t)$ and by performing the convolution along instantaneous streamlines (at time t') of the same field, $\mathbf{w}(\mathbf{x},t) = \mathbf{v}(\mathbf{x},t')$. The convolution is computed on a single spatial slice through I, i.e., $\mathscr{L}(s;\mathbf{x},t') = (\,\cdot\,,t')$. Unlike DLIC, the evolution of streamlines along pathlines might not lead to streamlines of the subsequent time step. In other words, the temporal evolution according to $\mathbf{u}(\mathbf{x},t)$ is, in general, not compatible with the structure of streamlines of $\mathbf{w}(\mathbf{x},t)$. For the special case of a steady flow, however, pathlines and streamlines are identical, and the time evolution is compatible with the construction of streamlines. As a solution, we propose to link the length of spatial patterns to the degree of unsteadiness. In regions where the flow changes rapidly, the correlated segments along streamlines should be very short or could even degenerate to point particles specified on I_i, while time-independent parts should be covered by long patterns. Note that the motion of unfiltered elements of I_i is, by construction, temporally correlated, regardless of the unsteadiness of the vector field. The framework allows for variable filter types and lengths, controlled by quantities derived from the input flow, and thus is prepared for spacetime-coherent flow visualization.

This visualization approach exhibits the following advantageous features. First, it shows both the motion of particles along pathlines and instantaneous streamlines in the same animation. Second, the degree of unsteadiness is revealed even in a single, frozen image in the form of the spatial frequency of noise along streamlines. Third, in the limit of a time-independent flow, the direction of motion is visualized similarly to animated LIC [Forssell and Cohen 1995]. UFAC cannot solve the fundamental dilemma of inconsistency between spatial and temporal patterns, but it explicitly addresses the problem and directly controls the length of the spatial structures. It maximizes the length of spatial patterns and the density of their representation, while retaining temporal coherence.

Other unsteady flow methods only implicitly address the issue, or neglect it completely. IBFV and LEA construct streaklines or time-reversed streaklines. When these lines intersect, the visualization may become convoluted or smeared out—it loses understandable spatial structure. An even more important problem of these techniques is the hysteresis in their visualization. An image strongly depends on the particle's past. In contrast, UFAC does not depend on visualization history, as long as particle injection by P_i is based on noise. Hysteresis is even more severe for UFLIC due to its feed-forward approach. Even for a steady flow, the structure of the display depends on how long the visualization has been running: the longer the time, the longer the correlation length along streamlines. Large-scale structures without high spatial frequencies in one direction can emerge and could therefore limit the spatial resolution perpendicular to the velocity field after this time-dependent flow has turned.

6 Implementation

Our implementation of the texture-based discretization of the framework is based on C++ and DirectX 9.0. GPU states and fragment programs (i.e., pixel shader programs) are configured within effect files. A change of this configuration can be included by changing the effect files without recompiling the C++ code. The source code of the essential effect files is provided on our project web page [Weiskopf 2003] to facilitate a re-implementation. Most parts of the framework can be readily mapped to the functionality of a DirectX 9.0 compliant GPU. A comparable implementation should be feasible with OpenGL and its fragment program support.

Any texture-related data is stored in 2D textures with fixed-point color channels; time dependency is represented by stacks of 2D textures. We start with the implementation of the first part of the framework: the construction of the property field. The vector field $\mathbf{u}$ is represented by fixed-point numbers in the red and green channels of a 2D texture by mapping the original floating-point components to the range $[0,1]$ (this mapping could be avoided by using floating-point textures). The two coordinates of a particle ($\tilde{\mathbf{X}}_l$), the accumulated property ($\tilde{I}_l$), and the accumulated weight are stored together in the RGBA channels of another 2D texture. The coordinates are chosen so that the range $[0.25,0.75]^2$ corresponds to the domain of the vector field; in this way, an area around the domain can be covered by valid fixed-point coordinates.

First, this texture is filled with the initial values by rendering a quadrilateral that covers the computational domain. The corresponding fragment program implements Eq. (6). The particle injection P_l is represented by a 2D texture with a single color channel for the brightness of the particle; the weight w is an outside program parameter. Then, the recurrence relations Eqs. (4) and (5) are evaluated by another fragment program that is executed by rendering a domain-filling quadrilateral Δi_τ times. This multipass approach makes use of the render-to-texture functionality of DirectX. The required values from $\tilde{I}_l$, P_l, and $\mathbf{u}$ are obtained by texture lookups, and the computations in Eqs. (3)–(5) are directly supported by the numerical operations in the fragment program. At last, the accumulated values $\tilde{I}_{l-\Delta i_\tau}$ are transferred to a single-channel representation of the final result by a trivial fragment program.

The second part of the framework—the convolution—can be mapped to graphics hardware in a very similar way. After each of these two parts, histogram equalization is applied to the accumulated values to restore a good contrast. A user-specified parameter makes possible a gradual change between full histogram equalization and no equalization. The histogram is built only once, by a CPU-based implementation. Equalization is applied to each fragment by a dependent texture lookup in the integrated histogram (i.e., in a color table). In addition, a Laplacian filter enhances the line-like patterns in the result of the final convolution; a 3×3 filter is employed in the current implementation. The strength of edge enhancement is also controlled by a user-set parameter. In addition, optional dye advection is implemented as a separate process similarly to [Weiskopf et al. 2001]; its results can be overlaid onto the noise-based visualization for the final display.

One important caveat is the resolution of texture channels. The accumulated property values are coded as 16 bits numbers to allow for an adequate accuracy for the weighted sum of a large number of samples along a path. For example, a filter length of 300 samples is quite common for the convolution step. Accuracy is even more important when histogram equalization with its non-uniform mapping is applied. We also use 16 bit channels to represent coordinate arrays. From our experience, these fixed-point numbers provide enough accuracy for the integration of particle paths and addressing texture coordinates. Even higher accuracy could be achieved by floating-point numbers. Conversely, particle injection textures such as input noise can be 8 bit or less. Finally, the vector fields are repre-

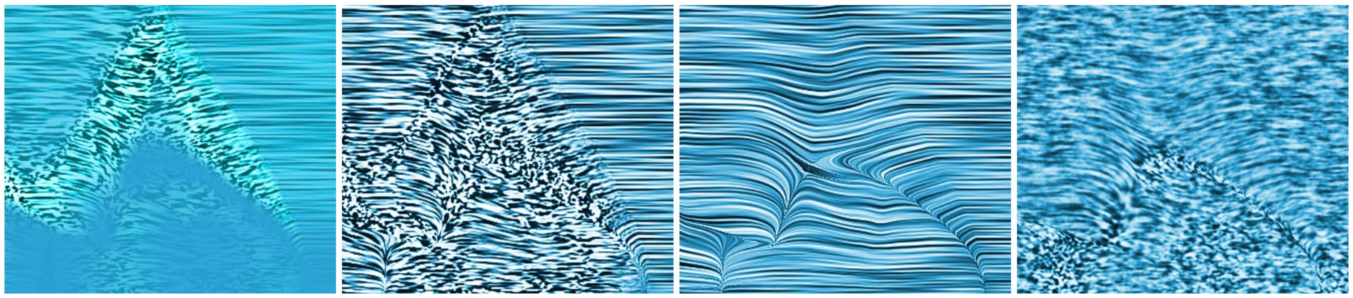
(a) (b) (c) (d)

Figure 3: Visualization of a shock data set, comparing UFAC with velocity masking (a), UFAC without masking (b), LIC (c), and LEA (d).

sented by fixed-point 16 bit numbers in the current implementation. Again, floating-point textures could be used for higher accuracy and range. However, bilinear interpolation is not supported by sampler stages for floating-point textures in DirectX 9.0 and must be reimplemented by computing the weighted sum of the four neighboring texels within a fragment program. On the other hand, bilinear interpolation is directly provided for fixed-point 8 or 16 bit textures.

7 Results

In this section, results obtained by the framework are presented and discussed. Videos contained on our project web page [Weiskopf 2003] clearly demonstrate the effects of temporal correlation through animations.

Figures 3(a)–(d) show the same snapshot from the animated visualization of a time-dependent shock data set from CFD. The complete data set has 200 time steps and a spatial resolution of 256×151. Figures 3(a) and 3(b) present UFAC. An additional velocity masking [Jobard et al. 2002] is applied to image (a) to emphasize regions of high velocity magnitude. In regions of large unsteadiness, such as the shock regions in the center of image, the correlation length along streamlines is reduced. In this way, unsteadiness is visualized even in a single image. Steady regions of the flow are represented by clearly defined, long streamline structures. Figure 3(c) is based on LIC, i.e., UFAC without controlling the length of streamlines. The important regions of unsteadiness are hard to recognize. Even worse, the animation exhibits substantial flickering in these regions because the temporal evolution is not compatible with the spatial structure of the streamlines. Figure 3(d) demonstrates the LEA implementation within the framework. LEA shows the magnitude of the velocity by the length of (inverse) streaklines, but tends to produce noisy images. The computational domain for all images is 512×302. For images (a)–(c), the maximum filter length is 120 and the maximum lifetime of particles is 40. The blending weight for LEA in Figure 3(d) is 0.05.

Figure 4 shows two frames of the visualization of a time-dependent electric field generated by an oscillating electric dipole. The evolution of field lines is governed by a velocity field specified in [Sundquist 2003]. The images were produced by the framework-based implementation of DLIC, on a resolution of 1024^2 with $\tau = 20$ and filter length 1200. This example shows that a high-quality and fast DLIC visualization is feasible on a purely texture-based representation.

Table 1: Performance measurements in fps for one iteration step.

Domain size	256^2	512^2	1024^2
Constructing I	5,100	1,380	350
Convolution	3,800	1,055	268

Figure 5 compares IBFV-guided techniques. Both images represent the very same underlying continuous model—with two different kinds of discretization. The same exponential filter kernel and the same accuracy of 16 bits for the property fields is used for both images. The computational domain has size 512^2. Figure 5(a) exhibits long, clearly defined streaklines generated by Lagrangian integration of trajectories in the framework-based approach. In contrast, Figure 5(b) shows strong blurring effects caused by the artificial diffusion in the original IBFV implementation (cf. the discussion in Section 5.2). This example is extreme because it uses only a few seed points and a filter with a long decay time. However, the blurring effect remains, to a smaller degree, even for shorter kernels and dense noise input.

The performance of the implementation depends mainly on the maximum lifetime of particles and the support of the convolution kernel. Table 1 shows the computation times of a single particle transport or convolution iteration (obtained by performing many steps and dividing by the number of steps). Times were measured on a PC with a Radeon 9700 GPU and an Athlon XP 2200+ CPU (1.8 GHz), running under Windows XP and DirectX 9.0. Convolution is slightly slower than constructing the property field because the fragment program is slightly more complex. The timings show an almost perfect linear dependency with respect to the number of texels in the domain. Furthermore, the complete visualization process depends linearly on the number of iteration steps. Consequently, the overall performance can be adjusted to a targeted frame rate by adapting the spatial resolution, filter length, and/or maximum lifetime, i.e., quality and speed can be smoothly balanced. In addition to the iterative computations, a small time offset is introduced by the general management of the visualization system (event handling etc.), edge enhancement, histogram equalization, velocity masking, color table mappings, final display to the screen, and transfer of time-dependent vector data from main memory to GPU. This transfer has only a minor effect on performance because just a single 2D texture for each new time step is passed to the GPU (since data for previous time steps has already been transferred). To give an impression of typical speeds for a complete visualization cycle that combines all these aspects: 12.7 fps are achieved for $\tau = 20$, filter length of 60, and a 512^2 domain; 3 fps for $\tau = 50$, filter length of 300, and a 512^2 domain.

8 Conclusion and Future Work

We have presented a generic framework for the visualization of time-dependent vector fields, along with a texture-based discretization that can be mapped to the SIMD architecture of graphics hardware for an efficient implementation. Spacetime-coherent dense representations are achieved 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 for spatially correlated patterns. The novel Unsteady Flow

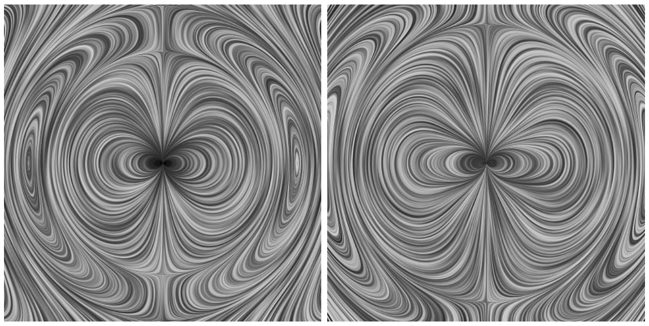
Figure 4: Electric fields of an oscillating electric dipole.

Advection–Convolution (UFAC) performs time evolution governed by pathlines, but builds spatial correlation according to instantaneous streamlines whose spatial extent is controlled to ensure their frame-to-frame coherence. In this way, UFAC specifically targets the visualization of unsteadiness. By an appropriate choice of parameters, the qualitative visualization results of LEA, IBFV, UFLIC, or DLIC can be obtained; since the underlying computational models may differ significantly, differences exist with respect to image quality, performance, and flexibility. In general, the framework allows the user to gradually balance visualization speed against quality.

UFAC and the other known techniques are only examples for specific choices of paths and filter kernels for the framework. We believe that more sophisticated visualization techniques are achievable within the same framework; for example, a more complex temporal dependency along the convolution paths or different types of spatial curves could be investigated. The final goal is a method that optimizes the parameters of the framework according to the input vector field (and maybe some other preferences). It is a challenge for future work to find appropriate quality measures and an optimization strategy. Considering a more basic and technical aspect, the dependency of the spatial frequency of noise transported along trajectories could be addressed by adapting the lifetime of particles according to the divergence or convergence of the vector field. Finally, the framework could be extended to visualizing 3D vector fields on curved hyperplanes and eventually to a completely 3D visualization.

Acknowledgments

The first author thanks Landesstiftung Baden-Württemberg for support, the second author acknowledges support from NSF under grant NSF-0083792.

References

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging vector fields using line integral convolution. In *Proc. ACM SIGGRAPH 93*, 263–272.

DIEWALD, U., PREUSSER, T., AND RUMPF, M. 2000. Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces. *IEEE Trans. Vis. Comput. Gr. 6*, 2, 139–149.

ERLEBACHER, G., JOBARD, B., AND WEISKOPF, D. 2003. Flow textures. In *Visualization Handbook*, C. R. Johnson and C. D. Hansen, Eds. Academic Press. In print.

FORSSELL, L. K., AND COHEN, S. D. 1995. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Trans. Vis. Comput. Gr. 1*, 2, 133–141.

HEIDRICH, W., WESTERMANN, R., SEIDEL, H.-P., AND ERTL, T. 1999. Applications of pixel textures in visualization and realistic image synthesis. In *ACM Symp. Interact. 3D Gr.*, 127–134.

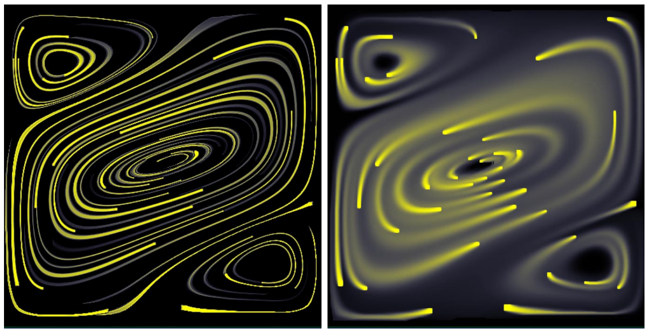
(a) (b)

Figure 5: Comparison of IBFV-guided techniques: long, clearly defined streaklines for the framework-based method (a); artificial diffusion for the original IBFV implementation (b).

JOBARD, B., AND LEFER, W. 2000. Unsteady flow visualization by animating evenly-spaced streamlines. In *Eurographics '00*, 31–40.

JOBARD, B., ERLEBACHER, G., AND HUSSAINI, M. Y. 2000. Hardware-accelerated texture advection for unsteady flow visualization. In *IEEE Vis. '00*, 155–162.

JOBARD, B., ERLEBACHER, G., AND HUSSAINI, M. Y. 2002. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Trans. Vis. Comput. Gr. 8*, 3, 211–222.

LIU, Z. P., AND MOORHEAD, R. J. 2002. AUFLIC: An accelerated algorithm for unsteady flow line integral convolution. In *EG / IEEE TCVG Symp. Vis. '02*, 43–52.

MAX, N., AND BECKER, B. 1995. Flow visualization using moving textures. In *Proc. ICASW/LaRC Symp. Vis. Time-Varying Data*, 77–87.

OKADA, A., AND KAO, D. 1997. Enhanced line integral convolution with flow feature detection. In *Proc. IS&T/SPIE Electr. Imag. '97*, 206–217.

SANNA, A., MONTRUCCHIO, B., AND MONTUSCHI, P. 2000. A survey on visualization of vector fields by texture-based methods. *Recent Res. Devel. Pattern Rec. 1*, 13–27.

SHEN, H.-W., AND KAO, D. L. 1998. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Trans. Vis. Comput. Gr. 4*, 2, 98–108.

SHEN, H.-W., JOHNSON, C. R., AND MA, K.-L. 1996. Visualizing vector fields using line integral convolution and dye advection. In *1996 Vol. Vis. Symp.*, 63–70.

STALLING, D., AND HEGE, H.-C. 1995. Fast and resolution independent line integral convolution. In *Proc. ACM SIGGRAPH 95*, 249–256.

SUNDQUIST, A. 2003. Dynamic line integral convolution for visualizing streamline evolution. *IEEE Trans. Vis. Comput. Gr. 9*, 3, 273–283.

VAN WIJK, J. J. 1991. Spot noise – texture synthesis for data visualization. *Comput. Gr. (Proc. ACM SIGGRAPH 91) 25*, 309–318.

VAN WIJK, J. J. 2002. Image based flow visualization. *ACM Trans. Gr. 21*, 3, 745–754.

VERMA, V., KAO, D., AND PANG, A. 1999. PLIC: Bridging the gap between streamlines and LIC. In *IEEE Vis. '99*, 341–348.

WEGENKITTL, R., GRÖLLER, E., AND PURGATHOFER, W. 1997. Animating flow fields: Rendering of oriented line integral convolution. In *Comput. Anim. '97*, 15–21.

WEISKOPF, D., HOPF, M., AND ERTL, T. 2001. Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations. In *Proc. VMV '01*, 439–446.

WEISKOPF, D., ERLEBACHER, G., HOPF, M., AND ERTL, T. 2002. Hardware-accelerated Lagrangian-Eulerian texture advection for 2D flow visualization. In *Proc. VMV '02*, 77–84.

WEISKOPF, D., 2003. Spacetime-coherent visualization of time-dependent vector fields. Web Site: http://www.vis.uni-stuttgart.de/ufac.