April 8, 2003

Flow Textures: High Resolution Flow Visualization

Gordon Erlebacher, Bruno Jobard, and Daniel Weiskopf

To appear in the *Handbook of Visualization*, Academic Press, late 2003.
Editors: Chris Johnson and Chuck Hansen

1 Introduction

Steady and unsteady vector fields are integral to many areas of scientific endeavor. They are generated by increasingly complex numerical simulations and measured by highly resolved experimental techniques. Datasets have grown in size and complexity motivating the development of a growing number of visualization techniques to better understand their spatio-temporal structure. As explained in Chapter ?? of this Handbook, they are often characterized by their integral curves, also known as pathlines. They can be best understood as the time evolution of massless particles released into the flow. In time-dependent flows, pathlines depend strongly on where the particles are released in space and time. Moreover, they can intersect when viewed in the physical domain, which often makes reliable interpretation of the flowfield quite difficult and prone to error.

Rather than visualize a collection of pathlines in a single slice, it is advantageous to consider instead the instantaneous structure of the flow, and its temporal evolution. For example, particles can be released along a planar curve and tracked. The time-dependent curve formed by the particles as they are convected by the flow is called a timeline. Visualization of streaklines, timesurfaces, etc. are other viable approaches based on integral curves. The extreme approach is to release a dense collection of particles, covering the physical domain, into the flow. At any point in time, due to flow divergence or strain, the particles form spatial patterns with varying degree of complexity, and their temporal evolution becomes of interest. These patterns may become non-uniform under the effect of flow divergence, convergence, or strain. A dense, if not uniform, coverage is maintained by tracking a sufficiently large number of particles. If a property is assigned to each particle, and if the number of particles is sufficiently large, one can construct

a continuous time-dependent distribution of the particle property over the physical domain.

Textures are a well-known graphic representation with many useful properties, and are well supported on graphics hardware since the mid 1990's. They execute a wide variety of operations, including filtering, compression, and blending, in hardware, and offer the potential to greatly accelerate many advanced visualization algorithms. Their main function is to encode detailed information without the need for large-scale polygonal models. In this paper, we discuss flow texture algorithms, which encode dense representations of time-dependent vector fields into textures, and evolve them in time. These algorithms have the distinctive property that the update mechanism for each texel is identical. They are ideally suited to modern graphics hardware that relies on a SIMD (Single Instruction Multiple Data) architecture. Two algorithms can be viewed as precursors to flow textures: spot noise [18] and moving textures [14]. Spot noise tracks a dense collection of particles, represented as discs of finite radius. These two algorithms motivated the development of dense methods and their eventual mapping onto graphics hardware. A key component of flow texture algorithms is a convolution operator that acts along some path in a noise texture. This approach to dense vector field representation was first applied to steady flows [5] and called line integral convolution (LIC). It has seen many extensions (Chapter ?? in this Handbook), some for unsteady flows, for example Unsteady Flow LIC (UFLIC) [16], Dynamic LIC (DLIC) [17], Lagrange-Euler Advection (LEA) [10], Image Based Flow Visualization (IBFV) [19], and others [14, 3]. Figure 1 illustrates the application of LEA to the evolution of unsteady vector fields on time surfaces [8].

Rather than enumerate existing algorithms along with their advantages and disadvantages, we present a conceptual framework within which these algorithms can be described and understood. We limit ourselves to 2D flowfields, although the strategies presented below have straightforward extensions to 3D, but at substantially higher cost.

The paper is structured as follows. Section 2 describes a new framework that abstracts the salient features of flow texture algorithms, in particular those related to temporal and spatial correlation. This is followed in Section 3 by a discussion of integration accuracy, texture generation, and the development of recurrence relations for easy mapping onto graphics hardware. Finally, we cover details necessary to port the leading algorithms on current graphics hardware in Section 4.

2 Underlying Model

Most of the existing flow texture algorithms derive from different approximations to a physical model that we now describe. Consider a single particle in an unsteady flow $\mathbf{u}(\mathbf{r}, t)$ and its *trajectory*, henceforth defined as a curve

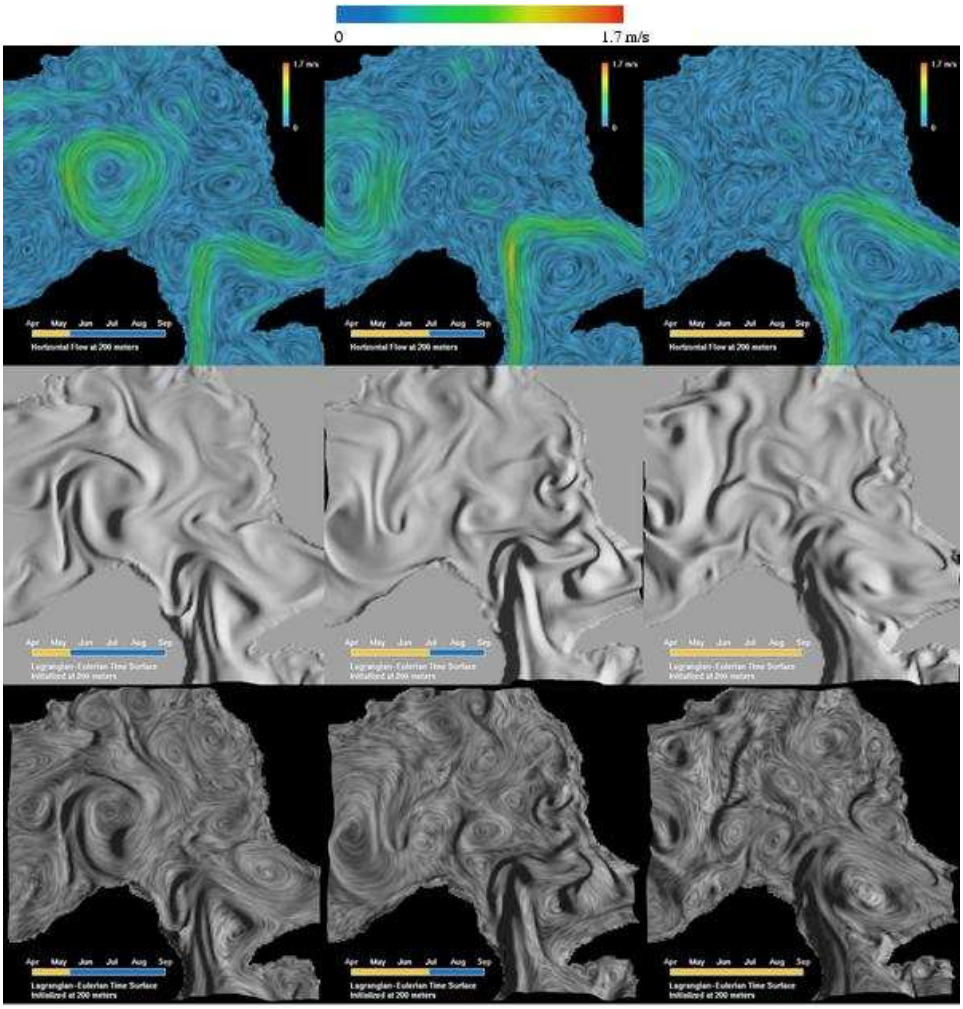

Figure 1: Application of flow textures to the advection of time surfaces in the Gulf of Mexico [8]. Top row: LEA, middle row: time surfaces viewed as a shaded surface, bottom row: flow texture superimposed on the time surface. Each row represents three frames from an animation. Data Courtesy J. O'Brien.

in a 3D coordinate system with two spatial axes $\mathbf{r}$ and a time axis t. As illustrated in Figure 2, the projection of the particle trajectory onto the (x, y) plane is a pathline of the underlying flow. The volume spanned by $\mathbf{r}$ and t is denoted by V. To understand the temporal dependence of the flow, the 3D reference frame is densely filled with particle trajectories. The intersection of the volume with a 2D spatial plane then yields a dense collection of points (also referred to as particles). As this slice translates along the time axis, their spatial coordinates change continuously, thus ensuring temporal coherence. Local regions of flow strain or divergence create pockets of increased and reduced particle density, which form distinct macroscopic patterns. These patterns persist in time when collections of neighboring particles travel together.

All methods must address the following challenges:

1. Maintain a time-independent particle density without destroying the information responsible for changes in the particle density and without sacrificing temporal coherence, i.e., particle trajectories must be sufficiently long.
2. Develop a good strategy for adding and removing particles.
3. Introduce spatial correlation into each frame that encodes information from a short time interval to generate macroscopic structures representative of the flow.
4. These macroscopic structures should be correlated in time to simulate flow motion.
5. Steady flow should result as a special case of unsteady flow.

2.1 Particle Features

Particles released into the flow are subject to several constraints. They are identified by an invariant tag, along with one or more time-dependent properties. In most implementations that target visualization, one of these properties is color, often kept constant. To accentuate particle individuality, the colors can have a random distribution. Alternatively, to simulate the release of dye into the flow, all particles initially within a local volume are assigned the same color. Particles are constrained to lie on a particle path imposed by an underlying vector field. Finally, particles can enter

and exit the physical domain, accumulate at sinks, or diverge from sources. Additional time-dependent properties can be assigned to particles, such as opacity. These might be used to enhance the clarity of any macroscopic patterns that may emerge.

At a given time, particles occupy a spatial slice through V and lie on separate trajectories that only intersect at critical points. The challenge is to develop strategies that maintain a uniform distribution of particles for all time. We address this next.

2.2 Maintaining a Regular and Dense Distribution of Particles

Under the above constraints, one is left with the task of generating particle trajectories that sample space uniformly and densely, and properly take into account effects of inflow/outflow boundaries. In a direct approach, a large number of particles is seeded randomly across a spatial slice and tracked in time. Particle coordinates are maintained in some continuously updated datastructure. Generally, one seeks to maximize the length of individual trajectories. However, this requirement must be balanced against the need for particle uniformity, which is easily destroyed as a result of boundary

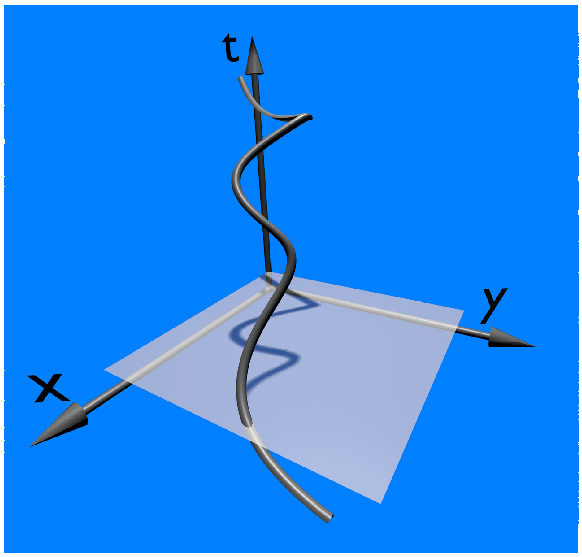

Figure 2: Illustration of a trajectory in the space-time domain V, and a pathline viewed as its projection onto a spatial slice.

effects and finite velocity gradients. Particles near inflow boundaries are transported into the domain leaving behind regions of reduced coverage. On the other hand, particles that exit the domain must be removed. Particle injection and destruction strategies are essential to counteract this effect. Control of particle density is also necessary within the interior of the physical domain.

We now present three strategies that achieve the above objectives, and relate them to existing flow texture algorithms.

1. All particles have a finite life span τ to ensure their removal from the system before non-homogeneities become too severe. A new distribution of particles is generated at constant time intervals. The density of particles is chosen to achieve a dense coverage of the spatial domain. This approach is at the heart of UFLIC [16] and time-dependent spot noise [6]. In UFLIC, the time between consecutive particle injections is smaller than τ. On the other hand, they are injected at intervals of τ for the spot noise approach. Aliasing errors are alleviated by distributing the initial particle injection in time.

2. Keeping the previous approach in mind, we now seek to explicitly control the density of particles. The spatial slice is partitioned into bins within which the particle density will remain approximately constant. Particles are removed or injected into a bin when their number exceeds or falls below prescribed thresholds. The length of trajectories is maximized by removing the oldest particles first. In this approach, a reduced number of particles is exchanged against more complex code logic. Accelerated UFLIC [13] and DLIC [17] are examples of this approach.

3. Particle tracking is abandoned in favor of an Eulerian point of view. The properties of a dense collection of particles on a spatial slice are sampled on a discrete grid. Their values are updated by following particle paths. This was the initial approach of Max and Becker [14]. More recent implementations are at the basis of LEA [10] and IBFV‘[19].

2.3 Temporal Correlation

To understand better the nature of temporal correlation, consider a single illuminated particle. This particle traces out a trajectory in space-time. Only a single dot is visible at any time. In the absence of temporal correlation,

the resulting image would be a set of dots distributed randomly. Temporal correlation is thus a consequence of the continuity of particle paths.

The human visual system [4] can integrate the smooth motion of a single bright dot and infer its direction and approximate speed. There is a range of speed that maximizes the effectiveness of this correlation. If the particle moves too rapidly, the visual system is incapable of correlating successive positions, and the trajectory information is lost. If the motion if too slow, information about the particle motion cannot be established.

Whatever the visualization technique used, the objective is to simultaneously represent the structure of the flow in a spatial slice through spatial correlation and the temporal evolution of these structures along particle trajectories. In the next section, we discuss a general framework to introduce flow patterns within a spatial slice. This general framework subsumes all known flow texture methodologies.

2.4 Spatial Correlation

To achieve this goal, consider an intensity function $I(\mathbf{r}, t)$ defined over the 3D space-time volume V introduced earlier. We define a filtered spatial slice

$$D_t(\mathbf{r}) = \int_{-\infty}^{\infty} k(s) I(\mathcal{Z}(t-s; \mathbf{r}, t)) ds \tag{1}$$

as a convolution along a trajectory $\mathcal{Z}(s; \mathbf{r}, t)$ through V. The subscript on D_t is a reminder that the filtered image depends on time. Trajectories are denoted by scripted variables and have three components: two spatial and one temporal. $\mathcal{Z}(s; \mathbf{r}, t)$ is parameterized by its first argument and passes through the point $(\mathbf{r}, t)$ when $s = t$. 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. It is necessary to introduce some correlation into V. To this end, we define the intensities in V to be either constant or piecewise continuous along another family of trajectories $\mathcal{Y}(s; \mathbf{r}, t)$. The structure of D_t is characterized by the triplet $[\mathcal{Y}, \mathcal{Z}, k]$.

Line Integral Convolution

LIC introduces coherence into a 2D random field through a convolution along the streamlines $\mathbf{x}(s; \mathbf{r}, t)$ of a steady vector field [5]. An equivalent formulation is possible within the 3D framework when $I(\mathbf{r}, t)$ is constant along lines parallel to the time axis with a random intensity across a spatial slice. Thus, $\mathcal{Y}(s; \mathbf{r}, t) = (\mathbf{r}, s)$ and $I(\mathbf{r}, t) = \phi(\mathbf{r})$, where $\phi(\mathbf{r}) \in [0, 1]$ is a random

function. The convolution is computed along $\mathcal{Z}(s;\mathbf{r},t) = (\mathbf{x}(s;\mathbf{r},t),s)$, i.e., the streamlines of the steady vector field. With these definitions, spatially correlated patterns produced by LIC are defined by

$$D_t(\mathbf{r}) = \int_{-\infty}^{\infty} k(s)\phi(\mathbf{x}(t-s;\mathbf{r},t))ds.$$

Image-Based Flow Visualization
IBFV [19] takes an approach similar to LIC with two exceptions. First, the convolution trajectories are based on the pathlines $\mathbf{x}(s;\mathbf{r},t)$ of $\mathbf{u}(\mathbf{r},t)$. Second, the intensity function defined in LIC is modulated according to a time-dependent periodic function $w(t)$ along the trajectories $\mathcal{Y}(s;\mathbf{r},t) = (\mathbf{r},s)$ that define $I(\mathbf{r},t)$. Thus the volume intensity distribution becomes

$$I(\mathbf{r},t) = w\left((t+\phi(\mathbf{r})) \bmod 1\right) \tag{2}$$

where $\phi(\mathbf{x}) \in [0,1]$ is now interpreted as a random phase. With the above definitions, the convolution is given by (1) with $\mathcal{Z}(s;\mathbf{r},t) = (\mathbf{x}(s;\mathbf{r},t),s)$.

Consider the case when a single trajectory from $\mathcal{Y}$ is illuminated with an intensity proportional to $w(t \bmod 1)$. The only points visible in $D_t(\mathbf{r})$ lie on trajectories that pass through $(\mathbf{r},s)$ for some s within the support of the kernel. When the kernel is a monotonically decreasing function of $|s|$, the segment of the curve that corresponds to $s < t$ is a streakline segment, with maximum intensity at $\mathbf{r}$, and decaying intensity away from $\mathbf{r}$. It should be noted that in the limit of steady flow, the resulting streamline intensity remains unsteady, unless $w(t)$ is constant.

Lagrangian-Eulerian Advection
LEA [10] seeks to model the effect of a photograph taken with a long exposure setting. Under such conditions, an illuminated particle forms a streak whose points are spatially correlated. This result is modeled by performing the convolution along trajectories $\mathcal{Z}(s;\mathbf{r},t) = (\mathbf{r},s)$ parallel to the time axis with constant intensity along the trajectories $\mathcal{Y}(s;\mathbf{r},t) = (\mathbf{x}(s;\mathbf{r},t),s)$ associated with $\mathbf{u}(\mathbf{r},t)$. The resulting display is then

$$D_t(\mathbf{r}) = \int_{-\infty}^{\infty} k(s)I(\mathbf{r},t-s)ds. \tag{3}$$

To identify the patterns formed by the spatial correlation, it is expedient to rewrite $D_t(\mathbf{r})$ as a convolution along some curve through a spatial slice of V. Since $I(\mathbf{r},t)$ is constant along pathlines of $\mathbf{u}(\mathbf{r},t)$, it follows that $I(\mathbf{x}(s;\mathbf{r},t),s) = I(\mathbf{x}(0;\mathbf{r},t),0)$ and

$$D_t(\mathbf{r}) = \int_{-\infty}^{\infty} k(s)I(\mathbf{x}(0;\mathbf{r},t-s),0)ds. \tag{4}$$

This is a convolution along the curve $\mathbf{x}(0;\mathbf{r},s)$ in the $t = 0$ slice and parametrized by s. If $k(s) = 0$ for $s > t$, all points on this path lie on trajectories that pass through $\mathbf{r}$ at some time $s \leq t$, which is a streakline of the flow $\mathbf{u}(\mathbf{r},t)$ after time reversal. The curves $\mathbf{x}(0;\mathbf{r},s)$, parameterized by s, are precisely the spatially correlated features we seek.

A comparison of equations (1) and (4) shows that IBFV and LEA are in a sense dual to each other. IBFV constructs V from lines parallel to the time axis and convolves along trajectories, while LEA constructs V from trajectories and does the convolution along lines parallel to the time axis. This duality is shown in Figure 3.

From the above discussion, we find that it is important to carefully distinguish between the curves along which the convolution is computed and the resulting patterns formed in $D_t(\mathbf{r})$. In general, the projection of $\mathcal{Z}(s;\mathbf{r},t)$ onto $D_t(\mathbf{r})$ differs from the resulting spatial patterns. If the kernel support is sufficiently small, the patterns are visually indistinguishable from short streamlines (streamlets). When $\mathbf{u}(\mathbf{r},t)$ is steady, both IBFV and LEA produce streamlines.

Dynamic LIC
Not all vector fields derive from the velocity field of a fluid. For example, electric fields driven by time-dependent electric charge distributions should be represented as a succession of LIC images correlated in time. In DLIC [17], the motion of the electric field $\mathbf{u}(\mathbf{r},t)$ is determined by a secondary flow field, $\mathbf{v}(\mathbf{r},t)$, with pathlines $\mathbf{y}(s;\mathbf{r},t)$. The generation of the

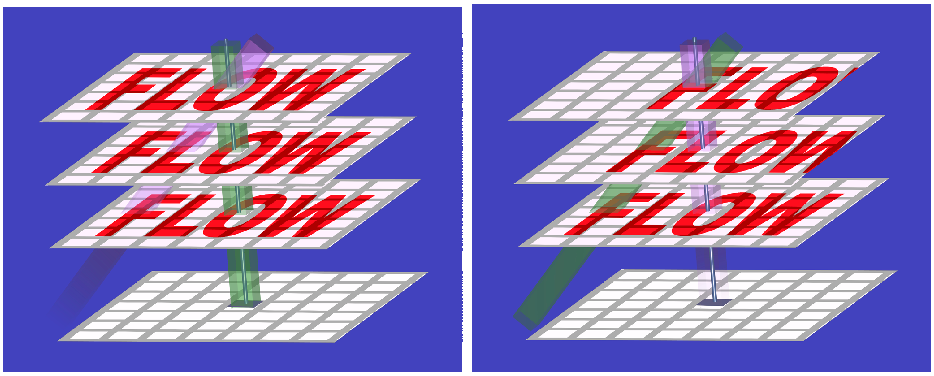

Figure 3: Duality between IBFV (left) and LEA (right). Volume intensity is constant along vertical lines and the convolution is along pathlines for IBFV. Intensity is constant along pathlines while convolution is along vertical lines for LEA.

sequence of LIC images is achieved by building $I(\mathbf{r}, t)$ from the pathlines of the motion field, i.e., $\mathcal{Y}(s; \mathbf{r}, t) = (\mathbf{y}(s; \mathbf{r}, t), s)$, and taking the convolution along the streamlines of $\mathbf{u}(\mathbf{r}, t)$, i.e., $\mathcal{Z}(s; \mathbf{r}, t) = (\mathbf{x}(s; \mathbf{r}, t), t)$. The filtered display becomes

$$D_t(\mathbf{r}) = \int_{-\infty}^{\infty} k(s) I(\mathcal{Z}(t - s; \mathbf{r}, t)) ds.$$

The resulting structures are streamlines of $\mathbf{u}(\mathbf{r}, t)$, transported in time along the pathlines of $\mathbf{v}(\mathbf{r}, t)$.

3 Implementing the Dense Set of Particles Model

Several ingredients are necessary to implement a flow texture algorithm: 1) the physical domain; 2) a time-dependent flowfield defined over the physical domain (and perhaps a second vector field to determine trajectories in non-flow simulations); 3) a mechanism to generate noise texture (suitably preprocessed); and 4) an integration scheme.

There are, several issues that have to be addressed in all methods: 1) accuracy; 2) sampling; 3) inflow boundaries; 4) uniformity of the noise frequency; and 5) contrast (see Section 4).

3.1 Integration Scheme and Accuracy

The position $\mathbf{x}(t)$ of a single particle subject to a velocity field $\mathbf{u}(\mathbf{r}, t)$ satisfies

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{u}(\mathbf{x}(t), t). \tag{5}$$

(In what follows, we no longer refer to the starting location along a path unless necessary for clarity.) In practical implementations of flow texture methods, the time axis is subdivided into uniform intervals Δt. Integration of (5) over the time interval $[t_k, t_{k+1}]$ yields the relation

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \int_{t_k}^{t_{k+1}} \mathbf{u}(\mathbf{x}(s), s) ds. \tag{6}$$

between particle positions. At the discrete level, the particle position at t_k becomes $\mathbf{p}_k = \mathbf{p}(t_k)$. A first order forward discretization of (6) relates the positions of a particle at times t_k and t_{k+1}:

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \Delta t\, \mathbf{u}(\mathbf{p}_k, t_k).$$

Similarly, a backward integration relates particle positions between times t_k and t_{k-1}:

$$\mathbf{p}_k = \mathbf{p}_{k+1} - \Delta t\, \mathbf{u}(\mathbf{p}_{k+1}, t_{k+1}).$$

A first order scheme is sufficient when integrating noise textures; errors only accumulate over the length of a correlated streak. These errors are in general sufficiently small so that they are not visible. When constructing long streaklines through dye advection (Figure 4, Section 3.3), this is no longer true. High order integration methods are then necessary to achieve accurate results [15, 10].

3.2 Sampling

The property field is sampled onto a texture that serves as a background grid. In a flow texture algorithm, it is necessary to update the value of the particle property on each cell (i.e., texel). The most general formulation is to compute a filtered spatial integration of the property at the previous time step over the physical domain, according to

$$C_t(\mathbf{r}_{ij}) = \int_{\text{physical domain}} K(\mathbf{r}_{ij} - \mathbf{r}) C_{t-\Delta t}\left(\mathbf{x}(t - \Delta t; \mathbf{r}, t)\right) d\mathbf{r} \qquad (7)$$

where $K(\mathbf{r})$ is some smoothing kernel. Different approximations to (7) lead to trade-offs between speed and quality. Many subsampling schemes, along

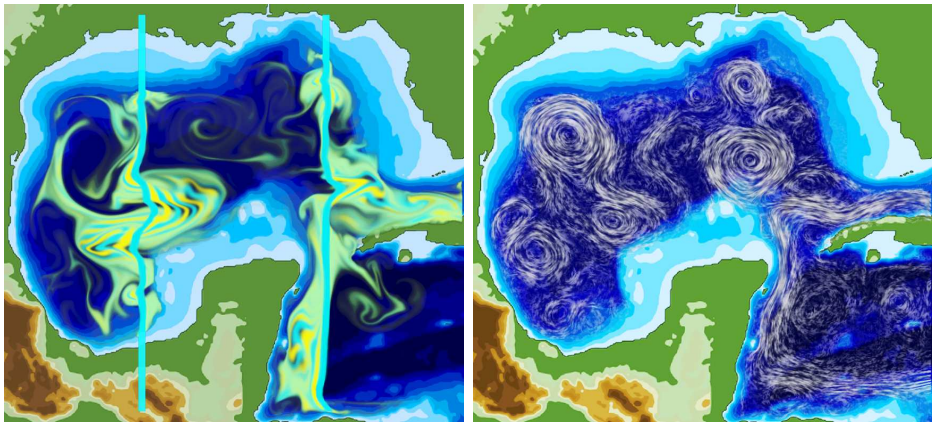

Figure 4: Flow in the Gulf of Mexico. Time lines visualized by dye advection (left), LEA with masking to emphasize regions of strong currents (right). Data Courtesy J. O'Brien.

with supporting theory, are described in [7]. The simplest approximation, and the fastest to execute, is point sampling, which takes the form

$$C_t(\mathbf{r}_{ij}) = C_{t-\Delta t}(\mathbf{r}_{ij} - \Delta\mathbf{r}_{ij}) \tag{8}$$

where

$$\Delta\mathbf{r}_{ij} = \int_{t-\Delta t}^{t} \mathbf{u}(\mathbf{x}(s; \mathbf{r}_{ij}, t), s)ds.$$

However, direct application of this formula may generate aliasing errors. LEA evaluates $C_{t-\Delta t}(\mathbf{r}_{ij} - \Delta\mathbf{r}_{ij})$ using nearest neighbor interpolation; all other methods use bilinear interpolation.

3.3 Texture Generation

Visual effects possible with flow texture algorithms strongly depend on the textures that are advected. Noise textures lead to dense representations of time-evolving streaks; dye advection results from overlaying a smooth background texture with local regions of color and letting these regions advect with the flow. Some techniques, such as IBFV, explicitly construct a time-dependent noise texture. In all cases, it is essential to pay attention to the quality of the textures generated to minimize aliasing artifacts produced by improper relationships between the properties of the velocity field, spatial and temporal sampling rates, and the frequency content of the input textures.

Texture for Noise Advection
We begin by illustrating the relationship between filtering and sampling through a simple 1D example. Consider a uniform velocity field u and a noise texture $I(x)$ filtered by convolution:

$$D(x) \quad = \quad \int_{-\infty}^{\infty} k(s)I(x - us)ds.$$

If $I(x)$ is sampled onto a uniform grid x_i with spacing Δx, a first order approximation of (9) yields

$$D(x) \quad \approx \quad \sum_{i=-\infty}^{\infty} k(i\Delta s)I(x - id)\Delta s,$$

where $I(x - id)$ is estimated from $I_i = I(x_i)$ via some reconstruction algorithm and Δs is the time sampling interval. By considering the properties of $D(x)$ when a single cell is illuminated, we derive a condition that relates Δs

and $d = u\Delta s$, the distance traveled by a particle over the sampling interval. Using the Heaviside function $H(x) = 1$ for $x > 0$ and zero elsewhere, the texture has the representation

$$I(x) = H(x + \Delta x) - H(x),$$

which leads to a filtered texture

$$D(x) = \sum_{i=-\infty}^{\infty} k(i\Delta s)[H(x + \Delta x - id) - H(x - id)]\Delta s.$$

We assume that the kernel is symmetric about $s = 0$. When $i = 0$, the term in brackets is a pulse of width Δx. When $i = 1$, the support of the pulse lies in $[d-\Delta x, d]$. To ensure overlap between these pulses it is necessary that Δx exceeds d. When this condition is violated, $D(x)$ is a series of disconnected pulses whose amplitudes follow $k(i\Delta x)$.

One way to avoid, or at least reduce, such aliasing effects is to ensure that high frequency components are removed from $I(x)$ by a pre-filtering step. The best filters have a short support in Fourier and physical space, making the Gaussian filter a strong candidate. Multifrequency filtering has been proposed as a means to link the characteristics of spatial patterns in $D_t(\mathbf{r})$ to properties of the flow, such as velocity magnitude [12]. In the context of flow textures, it has been shown that if a one-dimensional texture with a single illuminated texel is prefiltered with a triangular pulse of width greater than d, the resulting image is a smoothly decaying function that follows $k(s)$ [19]. The simplest 2D filters are isotropic or defined through a tensor product of 1D kernels. Spatial correlations introduced into the texture is then independent of orientation. Unfortunately, control is lost over the width of the streaks formed in the final image. Anisotropic filters can be designed to only prefilter the texture in the direction of the instantaneous flow field, while leaving the direction normal to the streamline untouched. None of the techniques addressed in this paper implement such an approach, although LEA does use a LIC algorithm as an anisotropic filter to remove aliasing in a postprocessing step. Glassner [7] provides a thorough discussion of issues related to sampling, filtering, and reconstruction.

Temporal Correlation

As explained in Section 2.4, $I(\mathbf{r}, t)$ is an intensity distribution defined over a space-time domain. It is important to ensure that the temporal sampling interval Δs is consistent with the frequency content of the volume. IBFV and LEA address this problem differently. In IBFV [19], the intensity map

is constructed independently of the underlying vector field. The temporal component is introduced through the periodic function $w(t \bmod 1)$ described by (2). High contrast is achieved when $w(t)$ varies sharply over some small t interval. To avoid aliasing effects, $w(t)$ should be filtered so that the time discretization satisfies the Nyquist criterion. Since $w(t)$ is applied to all points with different initial phase, the filtering should be performed analytically. Van Wijk [19] advocates the use of $w(t) = (1 - t) \bmod 1$, which emulates the effects of Oriented LIC [20].

Texture for Dye Advection
Dye advection is treated similarly to noise advection, although the objective is different. When advecting noise, we seek to visualize streak patterns representative of the magnitude and direction of vectors in the velocity field. Instead, dye advection techniques emulate the release of colored substance into a fluid and track its subsequent evolution over short or long time intervals. This process is simulated by assigning a connected region of the texture (referred to as dye) with a constant property value. When injecting dye, the texels located at the injection position are set to the color of the dye. Once released, the dye is advected with the flow.

3.4 Spatial Correlation

In this section, we discretize the convolution integrals (1) and (4) for IBFV and LEA respectively, and transform them into simple recurrence relations that can be implemented efficiently in hardware. A kernel that satisfies the multiplicative relation $k(s)k(t-s) = k(t)$ for $s \leq t$ leads to a recurrence relation between the filtered display at two consecutive time steps. The exponential filter $k(s) = \beta^{-1}e^{-\beta s}H(s)$, normalized to unity, satisfies this relation. Although such a filter does not have compact support, the ease of implementing exponential filters through blending operations makes them very popular.

Using the normalized exponential filter, Equation (1) is discretized according to

$$D_n(\mathbf{r}_{ij}) = \beta^{-1} \sum_{k=0}^{n} e^{-\beta s_k} \Delta s \, I(\mathcal{Z}(t_n - s_k; \mathbf{r}_{ij}, t_n)), \tag{9}$$

where the subscript n on D_n refers to t_n, $s_k = k\Delta s$, $t_n = n\Delta t$ and $t_0 = 0$. In general, $\Delta s = \Delta t$.

We next specialize the above relation to IBFV and LEA.

Image-Based Flow Visualization
IBFV uses trajectories

$$\mathcal{Z}(t_n - s_k; \mathbf{r}_{ij}, t_n) = (\mathbf{x}(t_n - s_k; \mathbf{r}_{ij}, t_n), t_n - s_k),$$

associated with $\mathbf{u}(\mathbf{r}, t)$ passing through the center $\mathbf{r}_{ij}$ of texel ij. Substitution into (9) yields

$$D_n(\mathbf{r}_{ij}) = \beta \Delta s \sum_{k=0}^{n} e^{-\beta s_k} I(\mathbf{x}(t_n - s_k; \mathbf{r}_{ij}, t_n), t_n - s_k). \tag{10}$$

Although IBFV defines $I(\mathbf{r}, t)$ through Equation (2), we derive a more general relationship valid for time-dependent intensity function. Using the relation $\mathbf{x}(s; \mathbf{r}_{ij}, t_n) = \mathbf{x}(s; \mathbf{r}_{ij} - \Delta\mathbf{r}_{ij}, t_{n-1})$ and some straightforward algebra, a simple relation emerges between $D_n(\mathbf{r}_{ij})$ and $D_{n-1}(\mathbf{r}_{ij} - \Delta\mathbf{r}_{ij})$, namely

$$D_n(\mathbf{r}_{ij}) = e^{-\beta \Delta t} D_{n-1}(\mathbf{r}_{ij} - \Delta\mathbf{r}_{ij}) + \beta \Delta s \, I(\mathbf{r}_{ij}, t_n). \tag{11}$$

Lagrangian-Eulerian Advection
In the case of LEA, $\mathcal{Z}(t_n - s_k; \mathbf{r}_{ij}, t_n) = (\mathbf{r}_{ij}, t_n - s_k)$. The discretization

$$D_n(\mathbf{r}_{ij}) = \beta \Delta s \sum_{k=0}^{n} e^{-\beta s_k} I(\mathbf{r}_{ij}, t_n - s_k), \tag{12}$$

of Equation (3) can be recast into the recurrence relation

$$D_n(\mathbf{r}_{ij}) = e^{-\beta \Delta s} D_{n-1}(\mathbf{r}_{ij}) + \beta \Delta s \, I(\mathbf{r}_{ij}, t_n). \tag{13}$$

The second term is evaluated by following the trajectory $\mathcal{Y}(s; \mathbf{r}_{ij}, t_n) = (\mathbf{x}(s; \mathbf{r}_{ij}, t_n), s)$ from $t = t_n$ to the previous time t_{n-1}, i.e.,

$$I(\mathbf{r}_{ij}, t_n) = I(\mathbf{r}_{ij} - \Delta\mathbf{r}_{ij}, t_{n-1}). \tag{14}$$

Dynamic Line Integral Convolution
As explained in Section 2.4, DLIC constructs an intensity map based on the pathlines of the motion field $\mathbf{v}(\mathbf{r}, t)$, updates it according to (14), and performs a LIC along the streamlines of $\mathbf{u}(\mathbf{x}, t)$. In the actual software implementation, a large number of individual particles, represented as discs of finite radius, is accurately tracked. The particles are allowed to overlap. Care is taken to ensure a uniform coverage of the physical domain, ensuring good temporal correlation. The final LIC is of high quality. Also note that LIC can be implemented in hardware [9].

3.5 Inflow Boundaries

All flow texture algorithms have difficulties with inflow boundaries, i.e., points along the boundary where the velocity points into the physical domain (Figure 5). The recurrence relations derived above for LEA and IBFV clarify the issues. In LEA, as evident from (13), particles that lie near an inflow boundary may lie outside the physical domain at the previous time. Should this occur, a random property value is assigned to that particle and blended into the dipslay at the current time. IBFV, on the other hand, must access the value of the spatially correlated display at $\mathbf{r}_{ij} - \Delta\mathbf{r}_{ij}$. Simply replacing this value by a random value would destroy the spatial correlation near the inflow boundaries, which would then contaminate the interior of the domain. For such points, one possible update equation is

$$D_n(\mathbf{r}_{ij}) = e^{-\beta\Delta t} D_{n-1}(\mathbf{r}_{ij}) + \beta\Delta s\, I(\mathbf{r}_{ij}, t_n).$$

4 Hardware Implementation

In this section, we demonstrate how the different flow visualization techniques described previously can be realized by exploiting graphics hardware.

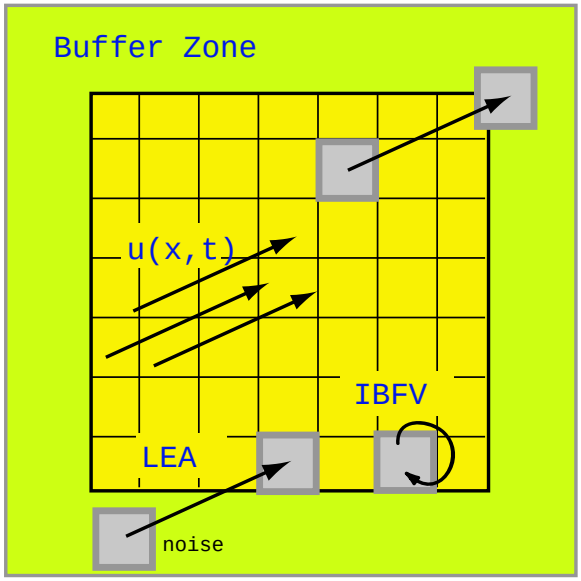

Figure 5: Inflow boundary treatment of LEA and IBFV.

The principal reason for using GPUs (graphics processing units) is to achieve a much higher processing speed, potentially two to three orders of magnitude higher than for a comparable CPU-based implementation. Performance is an important issue because it might make the difference between a real-time GPU-based visualization, which allows for effective user interaction, and a non-interactive CPU version. We start with an abstract view on the capabilities of GPUs and how visualization techniques can benefit, followed by a discussion of specific details of IBFV and LEA implementations.

Generic Hardware-Based Texture Advection
All implementations have to address the problem of how data structures and operations applied to them can be mapped to graphics hardware. From a generic point of view, the algorithmic structure of texture advection techniques consists of the following steps (see Figure 6). First, (noise) textures and possibly other auxiliary data structures are initialized. Then, each iteration step has to take into account: 1) the advection, based on the integration of pathlines, 2) a compositing operation, which combines information from the previous and the current time steps to introduce spatial coherence, 3) optional postprocessing to improve image quality, and 4) the final display on the screen.

The texture represents the particles that can be stored on the GPU by means of standard 2D or 3D textures, depending on the dimensionality of the domain. We assume that the particles are given on a uniform grid. In purely hardware-based implementations, the velocity field is also stored in a 2D or 3D texture. For hybrid CPU and GPU-based implementations, the vector field may be held in main memory and processed by the CPU. The advection step comprises both the integration of particle paths for one time step (based on (6)) and the transport of the texture along these paths (based on (8)). Typically, an explicit first order Euler integration is employed, which is executed either by the CPU for a hybrid approach or by the GPU. Texture transport is based on an appropriate specification of texture coordinates, which can be done either on a per-vertex basis (for hybrid CPU / GPU approaches) or on a per-fragment basis (i.e., purely GPU-based). The compositing operation is directly supported on the GPU by blending operations working on the framebuffer or within fragment programs.

Hardware-based implementations are very fast for the following reasons. GPUs realize a SIMD architecture, which allows efficient pipelining. In addition, the bandwidth to texture memory is very high, which leads to a fast texture access. Finally, a transfer of visualization results to the graphics board

for the final display is superfluous for GPU-based implementations. Since texture advection is compatible with the functionality of today's GPUs, a high overall visualization performance can be achieved.

However, the following issues have to be considered. First, the accuracy of GPUs usually is limited and might vary during the complete rendering pipeline. For example, color channels in the framebuffer or in textures have a typical resolution of eight bits, whereas fragment processing may take place at higher precision. Even floating-point accuracy within textures and fragment processing, provided by modern GPUs, is not comparable to double-precision numbers, available on CPUs. Second, the number of instructions might be limited. Therefore, the algorithms have to be designed to enable a rather concise implementation—sometimes at the cost of accuracy—or a less efficient multi-pass rendering technique has to be applied. Similarly, the number of indirection steps in fragment processing (i.e., so-called dependent texture lookups) is restricted. A third issue concerns the choice of APIs for programming the GPU. OpenGL [2] and DirectX [1], the two widespread

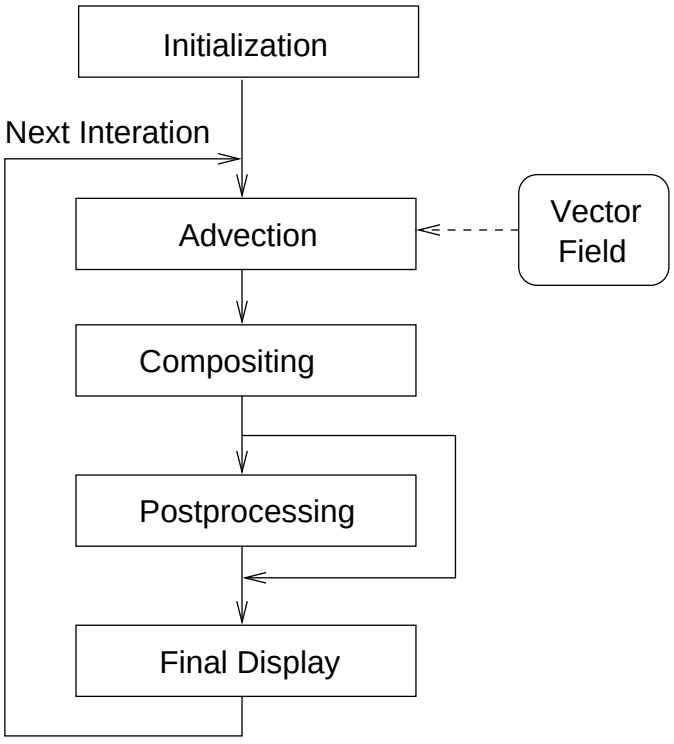

Figure 6: Flowchart for generic texture advection.

graphics APIs, have specific advantages and disadvantages. The main advantage of OpenGL is its platform (i.e., operating system) independence. However, at the time of writing, sophisticated features of the GPUs can only be addressed by GPU-specific OpenGL extensions. The situation might improve in the future with standardized interfaces for fragment programs. The main advantages of DirectX are a GPU-independent API and the support for a fast render-to-texture functionality, which is extremely useful for multi-pass rendering and iterative processes like texture advection. On the downside, DirectX is only available on the Windows platform. Finally, for vertex or fragment programming, higher-level programming languages, such as NVidia's CG (C for graphics) or the HLSL (high level shading language) of DirectX, can be used to replace the assembler-like programming with OpenGL extensions or DirectX vertex and pixel shader programs. We try to keep the following discussion as API-independent as possible in order to allow a mapping to current and future programming environments.

IBFV

Visualization techniques based on the idea of IBFV implement 2D flow visualization according to the recurrence relation

$$D_n(\mathbf{r}) = (1-\alpha)D_{n-1}(\mathbf{r} - \Delta t\,\mathbf{u}(\mathbf{r}, t_n)) + \alpha I_n(\mathbf{r}), \tag{15}$$

which approximates (11) by first order integration of the particle path and by first order approximation of the exponential function. Both D_n and I_n can be implemented by 2D textures. To alleviate the notation, D and I will henceforth refer to any of the D_n, I_n textures and $\mathbf{r}_{ij}$ is replaced by $\mathbf{r}$. The first term of (15) requires access to the texture D at the previous time step at the old particle position $\mathbf{r} - \Delta t\,\mathbf{u}(\mathbf{r}, t_n)$. Three possible solutions can be employed for this advection.

The first solution implements advection on a per-fragment basis. Here, a fragment program computes the old position $\mathbf{r} - \Delta t\,\mathbf{u}(\mathbf{r}, t_n)$ by accessing the vector field stored in another 2D texture. Then a lookup in D_{n-1} is performed by interpreting the old position as texture coordinates. This texture fetch operation is an example of a dependent texture lookup. Figure 7 shows the DirectX pixel shader code for this advection. The fragments are generated by rendering a single quadrilateral that spans the spatial domain. Texture coordinates t_0 and t_1 are specified in a way to exactly cover the domain by the textures. The dependent texture lookup takes the result of the previously fetched vector field $\mathbf{u}(\mathbf{r}, t_n)$, scales this result by a constant value (a parameter that is specified outside the fragment program and that takes into account $-\Delta t$), and adds the scaled value to the texture coordinates t_0

```
ps.1.0       // We are fine with pixel shader V1.0
tex t0       // Accesses vector field u
texbem t1,t0 // Dependent tex lookup in D with shifted tex coords
mov r0, t1;  // Outputs result
```

Figure 7: Pixel shader code for backward advection.

to obtain the coordinates for the lookup in the texture D_{n-1}.

The second solution implements a similar backward advection on a per-vertex basis [14]. The domain is covered by a mesh, with velocities assigned at its vertices. On the CPU, the old positions $\mathbf{r} - \Delta t\, \mathbf{u}(\mathbf{r}, t_n)$ are computed for the vertices by accessing the vector field in main memory. Once again, the old positions are interpreted as texture coordinates for a lookup in D. The domain is rasterized by drawing the entire mesh.

The third solution implements a forward advection on a per-vertex basis [19]. This technique differs from the previous one in that vertex coordinates are changed instead of the texture coordinates, i.e., a forward Euler integration $\mathbf{r} + \Delta t\, \mathbf{u}(\mathbf{r}, t_n)$ yields the vertex coordinates for a distorted triangle mesh. The results differ from backward integration because the Euler method is not symmetric with respect to the evolution of time. Nevertheless, first order forward and backward integration are both first order approximations of the true solutions, and forward mapping is an acceptable approximation to the advection in (15).

In all three implementations, the texture lookup with shifted texture coordinates makes use of bilinear interpolation. An interpolation scheme is required because pathline integration usually results in texture coordinates that do not have a one-to-one correspondence to the texels. The artificial smearing-out by bilinear interpolation does not invalidate the final images because their effects are continuously blended out by compositing according to (15).

This compositing operation can be achieved by a two-pass rendering [19]. The first pass writes the result of the above advection to the framebuffer. In the second pass, texture I is blended into the framebuffer by using α blending with weights α and $(1-\alpha)$, respectively. Alternatively, the advected texture D and I can be combined by multi-texturing within a fragment program. Finally, the framebuffer is saved in a texture to obtain the input for the subsequent iteration. In addition to noise-based visualization, IBFV emulates dye advection by interactively drawing additional dye sources into

the framebuffer during each rendering step.

LEA
Lagrangian-Eulerian advection is based on the recurrence relation (13), leading to

$$D_n(\mathbf{r}) = (1-\alpha)D_{n-1}(\mathbf{r}) + \alpha I_{n-1}(\mathbf{r} - \Delta t\, \mathbf{u}(\mathbf{r}, t_n)), \tag{16}$$

where I is computed from the property of the particle at the previous time step. When compared to IBFV, the roles of textures D and I are exchanged with respect to advection. A GPU-based algorithm [11, 21] implements a per-fragment advection of I analogously to the fragment-based backward mapping for D in the case of IBFV. However, since no α blending is imposed onto the transported noise texture I, a bilinear interpolation would cause an unacceptable smearing and a fast loss of both contrast and high frequency. Therefore, a nearest-neighbor sampling is employed during the backward integration step. Unfortunately, a direct implementation of nearest-neighbor sampling would not permit subtexel motion of particles because a texel is virtually re-positioned to the center of the respective cell after each iteration, i.e., small magnitudes of the velocity field would result in an erroneously still image [10]. The Lagrangian aspect of LEA makes possible subtexel motion: in addition to noise values, 2D coordinates of particles are stored in a texture; these coordinates are also updated during the particle integration and allow particles to eventually "jump" across texel boundaries even for small velocities. Additional discussions concerning numerical accuracy on GPUs can be found in [21].

Similarly to IBFV, textures D and I are combined by an α blending operation in the framebuffer or fragment program, and the framebuffer is saved in a texture to obtain the input for the subsequent iteration. Dye advection is included by a separate process that is based on per-fragment advection with bilinear interpolation. The final image is constructed from advected noise and dye textures by blending.

Postprocessing
The results of IBFV and LEA can be improved by postprocessing. Since both techniques apply a summation of different noise textures by blending, image contrast is reduced. Postprocessing by histogram equalization [19] or high-pass filtering [16] could be applied to increase contrast. Aliasing artifacts occur in LEA when the maximum texture displacement is excessive (see Section 3.3). These artifacts can be avoided by imposing LIC as a postprocessing filter [10]. As specifically designed for dye advection, artificial blurring caused by bilinear interpolation can be reduced by applying a filter

that steepens the fuzzy profile at the boundary of the dye, i.e., a non-linear mapping of the unfiltered gray-scale values [10]. Finally, velocity masking can be used to change the intensity or opacity depending on the magnitude of the underlying velocity field [10, 21]. In this way, important regions with high velocities are emphasized. Note that, with the exception of velocity masking [21], GPU-based implementations of the above postprocessing methods have not yet been reported in the literature.

5 Conclusions

We have presented the leading flow texture techniques within a single framework, first from a physical perspective, based on a space-time domain filled with a dense collection of particle trajectories. This was followed by a formulation that explains how to derive the visualization techniques based on properly chosen convolutions within the volume. We feel that flow texture algorithms are well understood in 2D planar flows, although some research remains to be done for flows on 2D curved manifolds. Flow texture algorithms in 3D remain a formidable challenge. Straightforward extensions to 2D algorithms are prohibitively expensive. The display of dense 3D data sets introduces several perceptual issues such as spatio-temporal coherence, depth perception, and orientation, that remain largely unsolved. Mechanisms for user interaction and navigation remain in their infancy. The holy grail of 3D flow texture algorithms is to develop a real-time system to display high quality dense representations with an interactively changing region of interest.

6 Acknowledgements

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

References

[1] DirectX. http://www.microsoft.com/directx.

[2] OpenGL. http://www.opengl.org.

[3] J. Becker and M. Rumpf. Visualization of time-dependent velocity fields by texture transport. In *Visualization in Scientific Computing '98*, pages 91–102, 1998.

[4] R. Blake and S.-H. Lee. Temporal structure in the input to vision can promote spatial grouping. In *BMCV 2000*, pages 635–653. Springer-Verlag, Berlin-Heidelberg, 2000.

[5] B. Cabral and L. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH 93*, pages 263–272, 1993.

[6] W. C. de Leeuw and R. van Liere. Spotting structure in complex time dependent flows. In *Scientific Visualization – Dagstuhl '97*. IEEE Computer Society Press, 1997.

[7] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufman, 1995.

[8] J. Grant, G. Erlebacher, and J. J. O'Brien. Case study: Visualization of thermoclines in the ocean using Lagrangian-Eulerian time surfaces. In *IEEE Visualization '02*, pages 529–532, 2002.

[9] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *ACM Symposium on Interactive 3D Graphics*, pages 127–134, 1999.

[10] B. Jobard, G. Erlebacher, and M. Hussaini. Lagrangian-Eulerian advection for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.

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

[12] M.-H. Kiu and D. C. Banks. Multi-frequency noise for LIC. In *Visualization '96*, pages 121–126, 1996.

[13] Z. Liu and R. Moorhead. AUFLIC - an accelerated algorithm for unsteady flow line integral convolution. In *EG / IEEE TCVG Symposium on Visualization '02*, pages 43–52, 2002.

[14] N. Max and B. Becker. Flow visualization using moving textures. In *Proceedings of the ICASE/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, 1995.

[15] H.-W. Shen, C. R. Johnson, and K.-L. Ma. Visualizing vector fields using line integral convolution and dye advection. In *1996 Volume Visualization Symposium*, pages 63–70, 1996.

[16] H.-W. Shen and D. L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, 1998.

[17] A. Sundquist. Dynamic line integral convolution for visualizing streamline evolution. *IEEE Transactions on Visualization and Computer Graphics*, 2003.

[18] J. J. van Wijk. Spot noise – texture synthesis for data visualization. *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25:309–318, July 1991.

[19] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

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

[21] D. Weiskopf, G. Erlebacher, M. Hopf, and T. Ertl. Hardware-accelerated Lagrangian-Eulerian texture advection for 2D flow visualization. In *Vision, Modeling, and Visualization VMV '02 Conference*, pages 439–446, 2002.