Visualization II Course Project
A Survey on Visualization of Time-Dependent Vector Fields by Texture-based Methods

Henry Derbes
Hderb001@odu.edu
757-348-4819

ODU Main Campus
May 7th, 2008
Abstract

The common goal of vector field visualization is to produce high resolution images that reveal flow field characteristics such as orientation, direction, magnitude and rate of change. The goal of this survey is to explore some of the fundamental ideas that led to the development of techniques for using texture and dye to visualize unsteady vector fields in two or three dimensions. Texture based methods enable users to visually resolve rather fine details of fluid flow such as small vortices.

This survey reviews some of the techniques for flow visualization that advance the practice beyond arrow plots and glyphs. Although arrow plots and glyphs can portray fluid vector field velocity, direction, acceleration, and can apply to three dimensional and time-dependent fields, the size of the area represented must be small due to the size of the icons. Particle tracing techniques offer an intuitive representation of transport along the flow. Dense representations for particle-tracing methods typically are built on texture-based techniques which provide images of high spatial resolution. This survey addresses some of the more significant developments in this area. Spot noise and Line Integral Convolution (LIC) offered early and quite useful methods of particle tracing techniques. LIC in particular was the basis for other techniques that sought to address limitations and offered improved performance such as dye advection, Oriented LIC (not discussed in this survey), Unsteady Flow LIC (UFLIC), and Dynamic LIC (DLIC). The survey explores texture advection techniques in which a dense collection of particles in a texture are transported according to the motion of particles in a time-dependent unsteady vector field. Two versions of the same ideas, Langranian-Eulerian Advection (LEA) and Image-Based Flow Visualization (IBFV) visualize unsteady flow by integrating particle positions and advecting the color of the particles based on a texture representation. In LEA, both noise and dye advection can be handled in the same framework. LEA can also be extended to three dimensions. In IBFV, the noise texture is advected and a second texture is blended into the advected texture at each time step. IBFV cannot be extended to three dimensions. Unsteady Flow Advection-Convolution (UFAC) offers the user control of spatial and temporal coherence by providing explicit and separate control of the patterns.
1. Introduction

Visualization of scalar and vector fields associated with flow over surfaces has many applications from the common scalar functions of two variables used in many disciplines to the three dimensional distribution of pressure and velocity over a ship hull or the wings of an aircraft [1]. The common goal of vector field visualization is to produce high resolution images that reveal flow field characteristics such as orientation, direction, magnitude and rate of change. Texture based methods enable users to visually resolve rather fine details of fluid flow such as small vortices.

The goal of this survey is to explore fundamental ideas that led to the development of techniques for using texture and dye to visualize unsteady vector fields in three dimensions. The author’s interest in this field is derived from a current work project in which the Department of Defense desires to better understand the consequences of weapon initiated explosions, the resulting unsteady vector fields, and the ability of structures, specifically ships, and people to survive the effects.

The fields of computational fluid dynamics and material engineering continue to challenge naval architects, meteorologists and aerospace engineers. Understanding the complex interactions of vessel and environment also continues to challenge those who wish to model and visualize fluid flow associated with ship and aircraft modeling with steady and unsteady vector fields. This survey reviews some of the early and more recent ideas about visualizing steady and unsteady vector fields using texture-based methods.

Most vector visualization algorithms depend on spatial resolution to represent the vector field. These include sampling the field with streamlines or particle traces or using icons at every vector field coordinate. The problem is that streamlines and particle tracing techniques depend critically on the placement of the “streamers” or particle sources. Depending on particle placement, eddies or currents in the data field can be missed. Icons do not miss data but use up a considerable amount of spatial resolution limiting their usefulness to small vector fields [2].

Time-dependent simulations of vector fields have become ubiquitous. A time-dependent method progressively tracks the visualization results over time. To achieve coherent animations, researchers developed a time-dependent method which can better characterize the evolution of the flow field by continuously tracking visualization objects such as particles [3].

Borrowing from Sanna et al [4], the following terms are used throughout this paper.

- Advection is the transport of a fluid.
- Convolution is a mathematical operator which takes two functions and produces a third function.
- Volume rendering creates a 2D image from scalar or vector datasets of multiple dimensions.
- Particle tracing techniques place a set of insertion points into a flow field. Particles are released from the insertion points to trace the flow pattern.
- Streamlines are tangent to a vector field at every point.
- Pathlines trace the trajectory of individual particles.
- Streaklines are the traces of a set of particles emitted from the same insertion points.
• Timelines link the particles emitted at the same time from different insertion points.

This paper is organized by the techniques developed for visualizing flow vector fields using textures. The first technique addressed in this survey, developed by van Wijk [1], is called Spot Noise and is described in section 2. In section 3, the Line Integral Convolution (LIC) by Cabral et al [2] is discussed along with a variation of the technique by Shen et al [5] for dye advection and animation. Another version of LIC called Unsteady Flow LIC (UFLIC) was developed by Shen [3] and is discussed in Section 4. UFLIC addresses temporal coherence in flow visualization. In section 5, we present Dynamic LIC, a technique that allows for the evolution of streamlines in time-dependent vector fields as developed by Sundquist [6]. Section 6 addresses Lagrangian-Eularian Advection (LEA) described by Jobard et al [7]. LEA visualizes unsteady flows by integrating particle positions and advecting the color of the particle based on texture representation. Section 7 includes a discussion of Image-based Flow Visualization (IBFV) by van Wijk [8], a method for 2D texture advection. In section 8, we discuss the Unsteady Flow Advection-Convolution (UFAC) technique developed by Weiskopf et al [9].

2. Spot Noise

In 1991, van Wijk was dissatisfied with previous methods of texture synthesis. They were not suitable because they did not provide local control of texture mapping. He proposed a technique that he called spot noise in which texture is synthesized by addition of randomly weighted and positioned spots. Van Wijk achieves local control of the texture by variation of the spot. The spot is a useful primitive for texture design, because in general, the relation between features of the spot and features of the texture are straightforward. Spot noise lends itself well for the synthesis of texture over curved surfaces [1].

Van Wijk [1] defines texture as the local variation in visual properties for the visualization of fields over surfaces. Motivated by previous research which had shown that the use of fixed patterns leads to poor results when compared to a stochastic model [1], van Wijk developed a convolution that includes a white noise texture. Texture synthesis occurs in two steps. First the data corresponding to texture coordinates is retrieved. Then the data is converted to parameter values using a mapping scheme. The mapping scheme expresses the variation in the data.

A grid of random values is generated representing white noise. The spot is a small intensity function over the data domain. Spot noise is synthesized through the convolution of a white noise grid and the spot.

Variation of the texture for data visualization is realized by variation of the spot. The variable spot \( h(p, x) \) has properties controlled by a set of parameters, \( p \). These parameters are determined via a data mapping \( m \) from the data \( d(x) \) that belong to the texture coordinates \( x \). Spot noise for data visualization can thus be synthesized by using variable spots:

\[
 f(x) = \sum_i a_i h(m(d(x_i)), x - x_i) \tag{1}
\]

The random scaling factor \( a_i \) has a mean of zero. The effect is that each spot represents a single particle dispersed in fluid flow and moving for a short time along a path. The shape of the intensity function \( h(m(d(x)), x - x_i) \) is
formed according to the vector field. This establishes local control. The sum of all spots characterizes the whole texture. This texture synthesis technique achieves clarity, ease of design and local control.

A disk is the simplest spot and the size affects the texture [1]. Figure 1 shows three disks with different radii and the corresponding textures. The differences in the textures of the bottom row can be explained from their power spectra.

![Figure 1. Spot size affects texture](image used by permission of the Association for Computing Machinery [1])

The disks work well for isotropic textures but the interesting parts of vector fields are anisotropic. Elliptical spots demonstrate anisotropic texture with the long axis scaled to reflect the data vector (figure 2) [1]. The pattern or shape of the spot also influences the texture (figures 3 and 4) [1].

![Figure 2. Non-proportional scaling of spots](image used by permission of the Association for Computing Machinery [1])

Figure 3. Regular patterns of the spots

(image used by permission of the Association for Computing Machinery [1])

Figure 4. Different shapes of the spots

(image used by permission of the Association for Computing Machinery [1])

Figure 5 shows four examples of the use of texture for visualization of scalar and vector fields. The colors indicate the value of a scalar field with saturated blue representing negative via grey to saturated red representing positive. In figure 5a, the variance of the texture indicates the absolute value. In figure 5b, gradients are emphasized by scaling the variance of the texture proportional to the norm of the gradient. In figure 5c, the scalar field is interpolated as a stream function $\psi(x,y)$ where $v_x = \frac{\partial \psi}{\partial y}$ and $v_y = \frac{\partial \psi}{\partial x}$. The flow velocity $\mathbf{v}$ is visualized by an elliptical spot with the major axis proportional to $|\mathbf{v}|$ and the minor axis proportional to $1/|\mathbf{v}|$. In figure 5d, the same principle is used but the scalar field is interpreted as a potential that defines the vector field $\mathbf{v}(x,y)$ as $v_x = \frac{\partial \psi}{\partial y}$ and $v_y = \frac{\partial \psi}{\partial x}$. 
Figure 5. Color denotes a scalar field

*(image used by permission of the Association for Computing Machinery [1])*

The standard texture mapping technique results in a distorted texture when mapped to a parametric surface (figure 6, left image). The yellow texture is mapped to the orange surface distorting along the vertical axis. In the right image, spot noise is easily distorted in texture space so that it is stationary in object space.

Figure 6. Mapping of texture to a parametric surface

*(image used by permission of the Association for Computing Machinery [1])*
Figure 7 shows a practical application of the techniques presented by van Wijk [1]. The image shows the visualization of the flow around a ship as calculated by the DAWSON simulation package by MARIN. MARIN is a large naval architecture company based in Norway. The colors and the white contour lines indicate the hydrodynamic pressure on the ship hull. Red denotes high pressure and blue low. The shape of the ship is visualized via shading and black equidistant cross-section lines. The flow velocity on the hull is visualized with spot noise. The spots are ellipses with the longest axis aligned with the flow direction. Texture distortion was used to compensate for the distortion caused by the parameterization of the hull’s surface.

![Figure 7. Visualization of velocity and pressure on a ship hull](image used by permission of the Association for Computing Machinery [1])

Spot noise allows easy local control and a wide variety of textures can be synthesized. The synthesis process is reasonably efficient. The use of texture means that some resolution is sacrificed. The largest scale of the texture has to be smaller than the scale of the variation in the data. In comparison to the use of color alone to display vector fields, texture has more degrees of freedom and can result in images that are more suggestive if not actually natural and realistic. Thus texture is probably more suited for global and qualitative visualization of data than for detailed and quantitative analysis.

Cabral [2] reviewed van Wijk’s [1] spot noise technique. The technique depends heavily on the form of the texture itself and does not generalize to other forms of textures that might be better suited to a particular class of vector data such as fluid flow versus electromagnetic field. Another weakness of spot noise is that when using elliptical spots, if the major axis exceeds the local length of the scale of the vector field, spot noise will inaccurately represent the vector field.
3. Line Integral Convolution (LIC)

Cabral [2] introduced the line integral convolution (LIC) algorithm. LIC advantages include: accuracy, locality of calculation, simplicity, controllability and generality. It also allows for the introduction of periodic motion filters important to signal and image processing.

LIC approximates the local behavior of the vector field by computing a streamline that starts at the center of pixel (x, y) and moves in the positive and negative directions of the vector. Only the directional component of the vector field is used in this advection. The magnitude of the vector field can be added in post processing steps.

With LIC, it is important to maintain symmetry about a cell. In the digital differential analyzer (DDA) spatial convolution technique defined by Cabral [2], each vector in the field is used to define a long narrow filter kernel that is tangential to the vector. A streamline is advected forward along a tangent to the vector field and backward by the negative direction of the vector field for some distance, L. The input texture is usually white noise data with the same resolution as the vector field. The input texture pixels under the filter kernel are summed, normalized by the length of the filter kernel, 2L, and placed in an output pixel image for the vector position. Figure 8 [2] illustrates this operation for a single vector in a field. This approach is efficient but inaccurate since it assumes that the local vector field can be approximated by a straight line.

Figure 8. Mapping of a vector onto a DDA line and input pixel field generating a single output pixel

(image used by permission of the Association for Computing Machinery [2])
The LIC technique offers a better approximation of the local behavior of the vector field by advecting forward to the next cell in the direction of the local vector (also backward in the negative direction). Continuous sections of the local streamline such as the straight line segments in figures 9 and 10, can be thought of as parameterized space curves in \( s \) and the input texture pixel mapped to a cell can be treated as a continuous scalar function of \( x \) and \( y \). It is then possible to integrate over this scalar field along each parameterized space curve. For each continuous segment, \( i \), an exact integral of a convolution kernel \( k(w) \) is computed and used as a weight in the LIC as shown in the equation:

\[
h_i = \int_{s_i}^{s_i + \Delta s_i} k(w) \, dw
\]

where \( s_0 = 0 \), \( s_i = s_{i-1} + \Delta s_{i-1} \), and \( \Delta s_i \) is the arc length between the points \( s_i \) and \( s_{i+1} \) along the streamline.

The entire LIC output pixel \( F_{\text{out}}(x, y) \) is given by the equation:

\[
F_{\text{out}}(x, y) = \sum_{i=1}^{l} \sum_{j=0}^{p} F_{\text{in}}(1, 0, 1) \frac{h_i + h_{i+1}}{h_i + h_{i+1}} F_{\text{in}}(0, 0, 1) h_i
\]

where \( F_{\text{in}}(1, 0, 1) \) is the input pixel corresponding to the vector at position \( (1, 0, 1) \), and \( l \) is such that \( s_i \leq L \leq s_{i+1} \), and primed variables indicate the negative direction counterparts to the positive direction variables. The numerator represents the line integral of the filter kernel times the input pixel field, \( F \). The denominator is the line integral of the convolution kernel and is used to normalize the output pixel weight.

The LIC algorithm visualizes local vector field tangents, but not their direction. If a periodic filter is used instead of a constant or box filter, then by changing the phase of the filters as a function of time, apparent motion in the direction of the vector field is created. Cabral [2] recommended creating a periodic low-pass filter to blur the underlying texture in the direction of the vector field. A Hanning filter, \( 1/\sqrt{2(1 + \cos(\theta + \phi))} \) has the desired property.

Figure 9. A 2D vector field showing local streamline starting in cell (x, y)
Figure 10. Forward coordinate advection

The LIC algorithm is designed as a function which maps an input vector field and texture to a filtered version of the input texture. The dimension of the output texture is that of the vector field. The input texture should be large enough so that the periodicity induced by coordinate wrapping is not apparent. Sanna [4] found several weaknesses with LIC.

1. Flow orientation is not displayed. The periodic filter with successive phase shifted images addresses this weakness.
2. Velocity magnitude cannot be inferred from the final output.
3. Only Cartesian grids can be handled.
4. The computational process is slow and real-time data exploration is not possible.
5. Only 2D cases can be considered, although Cabral [2] claimed that the LIC algorithm easily generalizes to higher dimensions.
6. Unsteady vector fields can be visualized only as a sequence of frames not time correlated.

Fluid flow experiments use external materials such as dye, hydrogen bubbles or heat energy injected into the flow to create stream lines, streak lines or path lines to highlight the flow patterns. Analogies to these experimental techniques have been adopted by scientific visualization researchers.

LIC methods successfully illustrate the global behavior of vector fields but provide no user probing capability. Specific information about the local behavior of the field is limited. Dye advection integrates local and global visualization techniques to explore 3D vector field data on regular grids. Using the LIC method as the underlying algorithm, Shen [5] developed an enhancement that enables the user to introduce “dyes” of various colors into the flow field. The dye propagates through the flow field highlighting local flow features such as wave fronts.
Dye injection is simulated by assigning colors to isolated local regions in the input white noise texture. Cells with streamlines that pass through such regions receive color contributions from the dye. In addition, phase shifting of the LIC algorithm can push the concentration of the dye along the streamlines. This creates the effect of dye propagation. To create a motion effect, the dye should smear only in the forward direction of the flow field. The dye should color only those cells whose negative streamlines pass through the dyed areas.

Initially, a regular LIC image, $F_{\text{out}}(x, y)$ for each animation step is computed using the white noise input. When the user injects the dye, the LIC convolution is applied using the dyed texture input to those cells that will be affected by the dye. The convolution is applied along these cell’s backward streamline directions to ensure that the dye only smears forward. The results are stored in $D_{\text{out}}(x, y)$ with the final image obtained by:

$$\text{Final}_{\text{out}}(x, y) = D_{\text{out}}(x, y) \otimes F_{\text{out}}(x, y)$$

Shen [5] proposed a fast searching method to rapidly locate those cells affected by the dye instead of running the complete LIC again. This is accomplished by storing the list of direct flow-back neighbors for each cell during the LIC algorithm. A root cell and its recursive direct flow-back neighbors constitute a directed graph.

For visualization of the dye advected vector field, Shen [5] used a bivariate volume rendering method. The first volume set is primarily data consisting of a volume of scalar variables representing the magnitude of the velocity or pressure or vorticity of the field. The opacity map of the primary data is manipulated in the same way as a standard volume rendering method to define the transparency of each voxel in the volume. The LIC texture serves as the secondary data and has its own color map but uses the transparency defined by the primary data. To render the two input data sets, Shen [5] defined a volume mixture to blend the two sets of data. This process gives the user control of a weighting parameter used in the blending process. As an additional control, the user can specify the opacity of the injected dye.

Shen et al [5] demonstrated their method (figures 11-13) in LIC images with dye propagation. Figure 11 has three different dye colors in a 2D vector field. The reader can observe the dye advecting in the vector field as the green and red disperse with the stream lines. Figure 12 is an animation sequence of dye advection in a 3D combustion simulation. Figure 13 is an animation sequence of dye advection in a tornado data set. The dye is colored based on the magnitude of the vector field.
4. Unsteady Flow LIC

Using LIC as the underlying approach, Shen [3] proposed a new convolution algorithm called Unsteady Flow Line Integral Convolution (UFLIC) to accurately model unsteady flow advection. The original method proposed by
Cabral [2] produces continuous flow textures that effectively illustrate global flow directions of a very dense flow field. It computes the traces of streamlines in a steady flow field but cannot be readily used for visualizing unsteady flow data.

The convolution algorithm proposed by Shen [3] simulates the advection of flow traces globally in unsteady flow fields. White noise is the input texture advected over time to create directional patterns of the flow at every time step. The advection is performed using a new convolution method called time-accurate value scattering scheme. In the time-accurate value scattering scheme, the image value at every pixel is scattered following the flow’s pathline trace which can be computed using numerical integration methods. At every integration step of the pathline, the image value from the source pixel is computed by collecting the deposits that have timestamps matching the current animation frame. To track the flow patterns over time and to maintain the coherence between animation frames, Shen [3] devised a process called successive feed-forward that drives the convolutions over time. The time-accurate value scattering process is repeated at every time step. It should be noted that each pixel scatters its value only in the forward pathline direction and not in the backward direction. Instead of using the white noise image as texture input every time, the resulting texture from the previous convolution step is used to compute the new convolution after performing high-pass filtering.

Image convolution can be thought of as a value gathering or value scattering scheme. In the value gathering scheme as proposed by Cabral [2], each pixel in the field travels in both positive and negative streamline directions gathering pixel values to compute the convolution. A different scheme, the value scattering scheme used in UFLIC, achieves the same result in a steady flow field. The value scattering scheme is implemented by first letting every pixel scatter its image intensity value along its streamline path. The convolution result for each pixel in the resulting image is computed by averaging the contributions made by the other pixels. Value gathering and value scattering produce equivalent results in a steady state vector field. However the results are different in time-varying vector fields.

In implementing UFLIC, Shen et al [3] used two notions of time to implement their method: physical time represented by \( t \), and computational time represented by \( \tau \). If \( t_i \) is the physical time at the \( i^{th} \) time step and if \( dt \) is the time between any two steps then \( t_{i+1} = t_i + dt \). Computational time at the \( i^{th} \) time step is represented by \( \tau_i \). The computational step size is one, so \( \tau_i = i \).

The time-accurate value scattering scheme computes a convolution image for each step of the time-varying flow data by advecting the input texture over time. The input texture starts with a white noise image and subsequently uses the convolution result output from the preceding step.

In UFLIC, given an input texture, every pixel in the field serves as a seed particle. From its pixel position at the starting physical time \( t \), the seed particle advects forward in space and time following a pathline defined as:

\[
p(t + \Delta t) = p(t) + \int_t^{t + \Delta t} \mathbf{v}(p(t), \tau) d\tau
\]

(5)
where $p(t)$ is the position of the particle at physical time $t$, and $p(t + \Delta t)$ is the new position after time $\Delta t$, and $v(p(t), t)$ is the velocity of the particle $p(t)$ at physical time $t$.

At every integration step, the input image value, $I_p$, of the pixel from which the particle originates is normalized and scattered to the pixels along the pathline. The normalization is determined by two factors: length of the current integration step and the “age” of the particle. For the first normalization factor, the distance between particle positions of the current and preceding integration is defined as $\omega$, expressed as:

$$\omega = ||p(t + \Delta t) - p(t)||.$$  \hfill (6)

The second factor simulates the particle’s fading intensity over time. The particle’s “age” at the $n$th integration step is:

$$A_n = \sum_{i=1}^{n-1} \Delta t_i.$$  \hfill (7)

The normalization value for the particle’s age is $\psi$ which decreases as “age” of the particle increases. If the particle’s lifespan is $T$ then:

$$\psi = 1 - \frac{A_n}{T}.$$  \hfill (8)

The overall normalization weight, $W$ is:

$$W = \omega * \psi.$$  \hfill (9)

The normalized scattering value at the $n$th integration step becomes:

$$I_{normalized} = I_p * W.$$  \hfill (10)

The normalized pixel value at every integration step is associated with a timestamp.

Each pixel has an associated buffer called the convolution buffer (C-buffer) to receive the scattering image values. Within the C-buffer are several buckets corresponding to different computational times. Each bucket accumulates image values and weights. The scattering at the nth integration step is done by adding the normalized image value $I_{accum}$ and its weight, $W_{accum}$, to the bucket associated with the pixel that corresponds to the computational time $\tau$. Each pixel’s final convolution value $C$ is computed from the values in the C-buffer associated with the current time:

$$C = \frac{I_{accum}}{W_{accum}}.$$  \hfill (11)

Shen [3] presented a time-dependent process called successive feed-forward that drives the time-accurate value scattering scheme to create temporal coherence. The initial input to the value scattering convolution algorithm is a regular white noise texture. The algorithm advects and convolves the noise texture to obtain the convolution result at the first time step. Subsequent convolutions use the output from the previous convolution as the input texture. The input texture with patterns formed by previous steps is then further advected. As a result the output frames in
consecutive time steps are highly coherent because the flow texture is continuously convolved and advected throughout space and time.

Shen [3] identified an issue with LIC in general and the value scattering scheme in particular. The methods acted as low-pass filtering process that will tend to diminish contrasts among flow lines over time as the same process is repeatedly applied to the input texture. Shen [3] applied a high-pass filter to the input texture before it is used by the value scattering scheme in the next step. The high-pass filter helped to enhance the flow lines and maintain the contrast in the input texture. Shen [3] used a Laplacian operator for the high-pass filter. To prevent the high-pass filter from introducing unnecessary high frequencies which might cause aliasing in the final image, Shen [3] jittered the resulting output from the high-pass filter with the original input noise texture. The resulting convolution image has restored contrast and clearer flow traces (figures 14 and 15). Figure 14 shows a snapshot from a sequence without the noise-jittered high-pass filter. Figure 15 is a snapshot in which the noise-jittered high-pass filter was employed.

![Figure 14. A convolution image generated from an input texture without noise jittered high-pass filtering [5]](image1)

![Figure 15. A convolution image generated from an input texture with noise-jittered high-pass filtering [5]](image2)

Jobard [7] discussed some of the difficulty with UFLIC while acknowledging that the algorithm achieves good spatial and temporal correlation. The difficulty lies in the interpretation of the images. The paths are blurred in regions of rapid change of direction and are thickest where the flow is nearly uniform. The high computing cost results from the large number of particles to process for each animation frame. The number of particles is three to five times the number of pixels in the image.

5. Dynamic LIC

Sundquist [6] described a technique that has the outstanding resolution of LIC but is able to generate animation sequences of time-varying fields with temporal coherence. Dynamic LIC (DLIC) extends LIC to time-dependent
fields making it possible to visualize the evolution of streamlines. The vector field varies arbitrarily over time with the motion of streamlines describes by a second “motion” vector field. Each frame is rendered using LIC. In DLIC, the input texture is generated by advecting a dense collection of particles over time and adjusting them to maintain the appropriate level of detail.

Sundquist [6] developed the DLIC method specifically for visualizing time-dependent electromagnetic fields. In most time-dependent flow methods, the motion of the field is along the direction of the field itself. For time-varying electromagnetic fields, this is no longer the case. The “motion” of the electric or magnetic field is independent of the field lines and generally not along the field lines. Unlike fluid flow models where the interest is in pathlines, in electromagnetism, the interest is in visualizing instantaneous field streamlines and how they evolve over time.

Sundquist [6] gave the problem formulation as a desire to visualize a dynamic vector field \( f \), with \( f \) being an arbitrary mapping from domain \( D \), and time interval \( T \) to two dimensional space. Field \( f \) evolves via a motion vector field \( d \), where the value of \( d \) at a particular point in space and time is the instantaneous velocity of the streamlines at the corresponding point in the vector field \( f \). The evolution the streamlines in \( f \) are prescribed in \( d \).

The DLIC algorithm first tracks a large number of particles as they evolve over time according to \( d \). Sundquist [6] called this the Particle Advection and Adjustment stage. The texture is adjusted by consolidating excess particles and adding new particles where needed. In the Texture Generation stage, these particles are then used to generate an input texture which is passed to a modified Fast LIC (FLIC) algorithm to produce the output.

The Particle Advection and Adjustment stage uses a white noise input texture. This helps ensure the output texture has high contrast perpendicular to the streamlines like that found in LIC images. To ensure correspondence between the streamlines evolving from frame to frame, DLIC warps the texture over successive time steps via the mapping defined by \( d \). This causes a loss of detail through repeated warping and filtering. Additionally, the motion field \( d \) may have divergent and convergent regions causing variations in detail. The texture mapping also falls apart at the edge of the domain \( D \) where the direction of motion given by \( d \) points inward. To avoid the problem, the algorithm continuously tracks particles and adjusts their distribution to keep the level of detail roughly the same in the output texture.

In the Texture Generation Stage, a frame of animation is generated by drawing all of the particles onto the output texture. Once the texture for a given frame is available, the FLIC method is applied to the texture to render the streamlines in the Fast Line Integral Convolution stage.

In the Particle Advection and Adjustment stage, a particle’s two-dimensional position, \( p_i \) is tracked with its intensity \( a_i \). Initially, a particle is created for each pixel and placed in the center but jittered slightly. Intensities are random, ensuring the first texture is white noise. Each successive frame is separated by time \( \Delta t \) and the particles are advected according to the motion field \( d \). After the particles have moved, the distribution of particles is assessed and adjusted as necessary. Particles that travel beyond the bounds of the texture are deleted first. Next, a texture coverage map, \( T_C \) is created. This map has the same resolution as the input texture and is used to determine how many particles are in each texture pixel. The texture coverage is the result of a sum of convolutions of the particles.
over the pixels and represents a fully general re-sampling of continuous particles onto the discrete texture grid (figure 16).

![Figure 16. Texture coverage [6]](image)

With the texture coverage, it is easy to find regions where particle density deviates significantly from the norm. Sundquist [6] limited the particle density to between 0.5 and 2.0 particles per pixel. If particle density is too low from either diverging field motion or because the texture does not have mapping into that region as occurs near some edges, the algorithm adds detail by creating a new particle with a random intensity. Although it may seem unreasonable to add detail where the field is diverging, it turns out that adding random particles is a very simple solution that gives satisfactory results without exhibiting any surprising appearance of new features in the diverging region.

The two operations of particle consolidation and particle creation succeed in maintaining a roughly uniform distribution of particles across the texture.

In the Texture Generation stage, the texture image itself at any point in time is calculated from the particles via a sum of convolutions of the particles scaled by their intensities over the pixels. This distribution of intensities in the texture varies with the density of the particles. In Sundquist's implementation [6], the particles have square profiles to simplify the computation. The algorithm to compute the texture simply iterates through the particles, summing the bilinear contributions scaled by the particle intensity to the four surrounding pixels (figure 17).

Once the texture for the current time step is generated, the FLIC technique is used to generate an image of the streamlines.
6. Lagrangian-Eulerian Advection

Jobard [7] proposed a technique to visualize dense representations of time-dependent vector fields using a Lagrangian-Eulerian Advection (LEA) scheme. Each still frame depicts the instantaneous structure of the flow, while an animated sequence of frames reveals the motion that a dense collection of particles would take when released into the flow. The algorithm produces animations with high spatio-temporal correlation at interactive frame rates. Combining the advantages of Lagrangian and Eulerian formalisms, the algorithm integrates particles backwards in time (the Lagrangian step), while the color distribution of the image pixels are updated in place (the Eulerian step). A few two dimensional arrays store all the information. The combination of Lagrangian and Eulerian updates is repeated at every iteration. A single time step is executed as a sequence of identical operations over all array elements. The algorithm takes advantage of spatial locality and instruction pipelining, generating animations at interactive frame rates.

With the LEA approach, Jobard [7] tracked a collection of particles $p_i$, along a prescribed time-dependent velocity field that densely covers a rectangular region. Property $P(p_i)$ is assigned to the $i$th particle, $p_i$. The property remains constant as the particle follows its pathline and has the role of a passive scalar. At any given instant $t$, each spatial location $x$ has an associated particle labeled $p_I(x)$. The particle property is invariant along a pathline and is expressed:

$$\frac{\partial P(p_i(x))}{\partial t} + \nabla \cdot P(p_i(x)) = 0$$

The trajectory of a single particle, denoted by $x^f(t)$ satisfies:

$$\frac{dx^f(t)}{dt} = \mathbf{v}^f(x^f, t)$$

In this Lagrangian approach, the trajectory of each particle is computed separately. In an Eulerian approach, particles lose their identity however, the particle property, viewed as a field, is known for all time at any spatial coordinate. Jobard [7] chose a hybrid method, the LEA method. Between successive time steps, coordinates of a dense collection of particles are updated with a Lagrangian scheme where as the advection of the particle property is achieved with an Eulerian method.
At the beginning of each iteration, a new dense collection of particles is selected, one particle per pixel. The particle’s position is integrated backward in time and it is assigned the property at that location as computed at the end of the previous iteration. This backwards position determines the color of the particle at its current location. Thus the core of the advection process is the composition of two basic operations, the coordinate integration and the property advection. Given the position $x^0(i,j) = (i,j)$ of each particle in the new image, backward integration of equation (13) over a time interval $h$ determines its position:

$$x^{-h}(i,j) = x^0(i,j) + \int_0^{-h} \nu^\tau(x^\tau(i,j))d\tau$$

(14)

at a previous time step. $h$ is the integration step, $x^\tau(i,j)$ represents intermediary positions along the pathline passing through $x^\tau(i,j)$, and $\nu^\tau$ is the vector field at time $\tau$.

An image of resolution $W \times H$, defined at a previous time $t - h$, is advected to time $t$ through:

$$I^t(i,j) = \begin{cases} I^{t-h}(x^{-h}(i,j)) & \forall x^{-h} \in [0,W) \times [0,H) \\ \text{user-specified value} & \text{otherwise} \end{cases}$$

(15)

which allows the image at time $t$ to be computed from the image at any prior time $t - h$.

The LEA algorithm requires manipulating $W \times H$ particles. All information concerning particles is stored in two-dimensional arrays with resolution $W \times H$ at the corresponding location, $(i,j)$. The initial coordinates $(x, y)$ are jittered and stored in arrays $C_x(i,j)$ and $C_y(i,j)$ where:

$$C_x(i,j) = i + \text{rand}[0,1] \quad \text{and} \quad C_y(i,j) = j + \text{rand}[0,1]$$

(16)

Arrays $C'_x$ and $C'_y$ contain the $x$ and $y$ coordinates after integration. Arrays $V_x^t$ and $V_y^t$ store the velocity field at the current time. Similar to LIC, LEA advects noise images. Four noise arrays $N, N', N_a, N_b$ contain respectively the noise to advect, two advected noise images, and the final blended image. $N_b$ is initialized with a two-valued noise function (0 or 1) to ensure maximum contrast and its values are copied to $N_b$.

A first order discretization of equation 14 is used to integrate the particle coordinates. After discretization with a constant time step $h$ over the entire domain, equation 14 becomes

$$\begin{cases} C'_x = C_x - \left(\frac{h}{V_{\text{max}}}\right)V_x(r_{\text{wr}}C_x, r_{\text{hr}}C_y) \\ C'_y = C_y - \left(\frac{h}{V_{\text{max}}}\right)V_y(r_{\text{wr}}C_x, r_{\text{hr}}C_y) \end{cases}$$

(17)

where $V_{\text{max}}$ is the maximum velocity magnitude in the whole data set, $r_{\text{wr}}$ and $r_{\text{hr}}$ are scaling factors ensuring that the coordinates of the velocity arrays stay within proper bounds.

The velocity arrays at the current time are linearly interpreted between the two closest available fields. Therefore, $h$ represents the maximal possible displacement of any particle over all iterations. The actual displacement of a particle is proportional to the local velocity and is measured in units of cell widths.

The advection of noise described in equation 15 is applied twice to $N$ to produce noise arrays $N'$ for advection, and $N_a$ for display. $N'$ is an internal noise array with the purpose of carrying forward the advection process and to
re-initialize \( N \) for the next iteration. \( N' \) is computed with a constant interpolation to maintain sufficiently high contrast in the advected noise. Before \( N' \) can be used in the next iteration, it must undergo a series of noise corrections to account for edge effects, the presence of arbitrary domains, and the deleterious consequences of flow divergence.

\( N_a \) serves to create the current animation frame and no longer participates in noise advection. It is blended into \( N_b \) as discussed below.

Jobard [7] provides an efficient implementation that eliminates the need to test for boundary conditions. The implementation adds a buffer zone of constant width \( b = \lfloor h \rfloor \) (figure 18). The expanded array of size \((W + 2b) \times (H + 2b)\) ensures that out of bounds array accesses do not occur.

![Figure 18. Noise arrays expanded with buffer b cells wide [7]](image)

A common issue with texture advection is the proper treatment of information flowing into the physical domain. In LEA, the advected image contains a two-valued random noise with little or no spatial correlation. The method simply stores random value (0 or 1) noise in the buffer zone at negligible cost. At the next iteration, \( N \) will contain these values with some values advected to the interior of the physical domain.

Some vector fields are not rectangular and some contain interior regions where flow is not defined. The reader may think of a shore or island in an ocean current. LEA handles this with no modification by simply setting velocity to zero where it is not defined. The stationary noise is hidden from the animation frame by superimposing a semitransparent map that is opaque where the flow is undefined.

Coordinate arrays are reinitialized to prepare a new collection of particles to be integrated backward in time for the next iteration. Particle coordinates are not reinitialized to their initial values. Advection of particles with small velocity magnitude results in intra-cell displacement and no change in property. If the particles are initialized at their original positions, the flow would appear frozen where the velocity magnitude is too low as in figure 19a. Constant interpolation without fractional coordinate tracking clearly shows that the flow is partitioned into distinct regions within which the integer displacement vector is constant.
In Figure 19b, the fractional part of the displacement within each cell is tracked and added to the coordinate initial values:

\[
\begin{align*}
(c_x(i,j)) &= i + \frac{\partial_x}{\partial}(i,j) - \frac{\partial_x}{\partial}(i,j) \\
(c_y(i,j)) &= i + \frac{\partial_y}{\partial}(i,j) - \frac{\partial_y}{\partial}(i,j)
\end{align*}
\]

Although successive advected noise arrays are correlated in time, each individual frame remains devoid of spatial correlation. Jobard [7] applied a temporal filter to successive frames to achieve spatial correlation. An exponential filter is easy to implement through an alpha blending of the current advected noise, \(N_\alpha\) and the previous filtered frame, \(N_b\): \(N_b = (1 - \alpha)N_b + \alpha N_\alpha\), where \(\alpha\) represents the opacity of the current noise advected array. The blending stage is crucial because it introduces spatial correlation along pathline segments in every frame.

The filtering phase completes one pass of the advection algorithm. The image \(N_b\) can be displayed to the screen or stored as an animation frame. \(N'\) is used as the initial noise texture \(N\) for the next iteration.

Weiskopf [9] offered some disadvantages of LEA. LEA exhibits suboptimal temporal correlation which becomes apparent in the form of noisy and rather short spatial patterns after convolution. Another disadvantage is that LEA is limited to white noise input textures.

7. Image-based Flow Visualization (IBFV)

Van Wijk [8] presented a method for visualization of two-dimensional vector fields. Van Wijk provided a single framework to generate a wide variety of visualizations of flow from traditional particle tracing and streamlines, to moving textures and topological images. Three other features of the method are: handling of unsteady flow, efficiency and ease of implementation.

A general description of the method is simple in concept. Each image is the result of warping the previous image in response to a vector field, followed by blending with some background image. The process is accelerated by taking advantage of graphics hardware. The combination of these two approaches had not been presented before and van Wijk [8] claimed this method is close to optimal. A major contribution of IBFV is the definition of the
background images that are blended, eliminating the need for post processing to eliminate high frequency components. Van Wijk [8] touted IBFV by providing a general evolution of visualization. "Flow visualization methods like particle and streamline tracking operate in world space, and have geometric objects like points and lines and primitives. The Line Integral Convolution method starts in screen space: For each pixel, a streamline is traced. (IBFV) takes complete images as the basic primitive. This leads to straightforward definitions and algorithms, which can be mapped effectively on graphics hardware." (p. 747)

Image generation: Assume that an unsteady two-dimensional vector field, \( \mathbf{v}(x; t) \in \mathbb{R}^2 \) where
\[
\mathbf{v}(x; t) = [v_x(x, y; t), v_y(x, y; t)]
\]
has been defined for \( t \geq 0 \) and for \( x \in S \), where \( S \subset \mathbb{R}^2 \). A pathline is obtained by tracking the position of a particle in a dynamic flow field. A pathline is the solution of the differential equation
\[
\frac{dp(t)}{dt} = \mathbf{v}(p(t); t)
\]
for a given start position \( p(0) \). A first order approximation of equation (20) gives the Eulerian integration method
\[
p_{k+1} = p_k + \mathbf{v}(p(t); t)\Delta t
\]
with \( k \in N \) and \( t = k\Delta t \). Here the frame number \( k \) is a unit of time.

Suppose a field \( F(x; k) \) represents some property such as an image advected by the flow. A property is advected just like a particle, so \( F(p(t); t) \) is constant along pathline \( p(t) \). A first order approximation of the transport of \( F \) is given by:
\[
F(p_{k+1}; k + 1) = \begin{cases} 
F(p_k; k) & \text{if } p_k \in S \\
0 & \text{otherwise} 
\end{cases}
\]
Van Wijk [8] sets \( F(p_{k+1}; k + 1) \) to 0 (black) if the velocity at a previous point \( p_k \) is undefined. Any pathline with a starting point \( p_0 \) outside the image \( S \) will be black. Sooner or later most of \( F(x; k) \) will be black. To remedy this at each time step, van Wijk [8] creates a convex combination of \( F \) and another image \( G \). Specifically:
\[
F(p_{k}; k) = (1 - \alpha)F(p_{k-1}; k - 1) + \alpha G(p_{k}; k)
\]
where \( \alpha(x, k) \in [0,1] \) is a blending mask. For many applications \( \alpha \) is constant in time and space. Equation (23) defines the image generation process and forms a start point for analysis. Eliminating the recurrency term gives:
\[
F(p_{k}; k) = (1 - \alpha)^k F(p_0; 0) + \alpha \sum_{i=0}^{k-1} (1 - \alpha)^i G(p_{k-i}; k - i)
\]
The first term represents the contribution of the first image and can be ignored if the first image is black or if \( k \) is large. Hence we get:
\[
F(p_{k}; k) = \alpha \sum_{i=0}^{k-1} (1 - \alpha)^i G(p_{k-i}; k - i)
\]
So for a large enough \( k \), the color of a point \( p_k \) of the image is the result of a line integral convolution of a sequence of images \( G(x; t) \), along a pathline through \( p_k \), with an exponential decay convolution filter, \( \alpha (1 - \alpha)^i \). Note that a
low $\alpha$ gives a high correlation along the pathline. This is an Eulerian point of view since the user observes what happens at a point. From a Langrangian perspective, we consider what happens as a particle moves along. Particles are advected by the flow and fade away exponentially.

Van Wijk [8] uses random noise images for $G$ and develops masks $\alpha$ to control the image generated. He [8] also proposes a triangular pulse for $g(x)$. Let function $h_w(x)$ represent a triangular pulse where:

$$h_w(x) = \begin{cases} 1 - |x|/w & \text{if } |x| \leq w \\ 0 & \text{otherwise} \end{cases}$$

and let $g(x) = h_w(x)$. For a small $w$, the corresponding $f(x)$ is a series of triangles with constant width and exponentially decreasing height. Overlapping triangles are avoided if $w \geq |v|\Delta t$.

IBFV uses a low-pass filter of a linearly interpolated sequence of $N$ random values, $G_i$ with $i = 0, 1, \ldots, N-1$, and

$$g(x) = \sum h_s(x - is)G_{i \mod N}$$

where the spacing $s$ satisfies $s \geq \Delta t$. To satisfy this constraint, a large value of $s$ can be used or the velocity field can be modified by clamping the velocities to an upper limit $v_{\text{max}}$ with $v_{\text{max}}\Delta t \leq s$. In figure 20, a velocity field is modeled using the background noise in the top image. The second image is a vector field with a large value of $s$. There is a source and a saddle point in the image. Texture gives an indication of velocity direction and magnitude. The saddle point has a velocity of 0 while the greatest velocity is near the source. The third image used a low value for $v_{\text{max}}$ and a large value of $\Delta t$, which has the effect of clamping most of the velocity field. The result is an image that indicates direction only but no velocity information. In the bottom image, the artifacts are the result of high $v_{\text{max}}$ and a large value of $\Delta t$.

![Figure 20. A source in a linear flow](image used by permission of the Association for Computing Machinery [8])
To produce animated images, van Wijk [8] used a set of $M$ images as background noise, linearly interpolated in space and discrete in time:

$$g(x; k) = \sum h_k(x-\varepsilon s)G_{k \mod M \mod M}$$

where $k$ is the frame number. The simplest solution is to use a different set of random values for $G_{ik}$, however the variation along a pathline is too strong. Van Wijk [8] used a version of the spot noise technique in which the intensity of the spot varies as it moves. A saw tooth pattern for varying the intensity seems to work best.

A common problem of flow visualization using texture mapping is reduced contrast. Van Wijk [8] found the results of post processing disappointing. The images without post processing seem to be sufficient when used as background for dye-based techniques. Boundary areas are another common problem for vector field visualization. Van Wijk [8] defined boundary areas $B = S - S'$ where $S$ is the original flow domain and $S'$ is the distorted flow domain (figure 21).

![Figure 21. Boundary Area $B = S - S'$](image used by permission of the Association for Computing Machinery [8])

Van Wijk [8] used a modified version of equation (23) to handle texture in the boundary area:

$$F(p_k; k) = (1 - \alpha)F^*(p_{k-1}; k-1) + \alpha G(p_k; k) \text{ with } (29)$$

$$F^*(p_k; k-1) = \begin{cases} F(p_k; k-1) & \text{if } p_k \in B \\ F(p_{k-1}; k-1) & \text{otherwise.} \end{cases}$$

The boundary area $B$ is not advected, just constantly blended with new values of $g$.

Dye advection fits well in the IBFV model. Injection of dye with color $G_D$ and blending mask $\alpha_D$ to an image $F$ can be modeled as:

$$F'(p_k; k) = (1 - \alpha_D(x; k))F(x; k) + \alpha_D(x; k)G_D(x; k) \text{ with } (30)$$

The final modified image $F'(p_k; k)$ is passed to the advection step next where the injected dye is advected.

8. Unsteady Flow Advection-Convolution (UFAC)
In developing UFAC, Weiskopf [9] addressed the two types of coherence that are essential for understanding animated visualizations. Spatial coherence reveals some vector structure within a single picture. Temporal coherence allows the user to identify a consistent motion of the changing structure by achieving frame to frame coherence.

UFAC provides the user with separate control over the two types of coherence through a generic and flexible framework that provides spacetime coherent dense representation of time-dependent vector fields by a two-step process. Weiskopf [9] defined spacetime as a three dimensional space with two special dimensions and one temporal dimension. First, continuous trajectories are constructed in spacetime to guarantee temporal coherence. Second, convolutions along another set of paths through the above spacetime result in spatially correlated patterns. The framework can be mapped to a discrete texture-based representation in combination with a single instruction multiple data (SIMD) architecture to make an efficient implementation on graphics hardware. The framework makes use of two different sets of paths to achieve temporal and spatial coherence. This method allows use of more advanced visualization techniques. The UFAC method performs time evolution of unsteady fluid flows using pathlines, but builds spatial correlation according to instantaneous streamlines. The spatial extent of the streamlines is related to the degree of unsteadiness of the vector field.

Weiskopf [9] used 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. Properties such as a grey scale value [0, 1] are attached to the particles to distinguish them from one another. Properties are allowed to change continuously along the trajectory however, they often remain constant.

A function, \( I(x, t) \) is a property field defined by setting its value to the property of a particle at a spacetime point, \( (x, t) \). The continuous behavior of trajectories and their attached properties ensures that spatial slices through the property field at nearby times are strongly related achieving temporal coherence.

In general, different particles are not correlated and spatial slices of the property field do not exhibit any coherent spatial structures. To achieve spatial correlation, a filtered spatial slice \( D_{r} (x) \) is defined through the convolution:

\[
D_{r} (x) = \int_{-\infty}^{\infty} k(s) I \left( \mathcal{T}(t - s; x, t) \right) ds
\]  

(31)

along a path \( \mathcal{T}(s; x, t) \) through spacetime. The subscript on \( D_{r} \) is a reminder that the filtered image depends on time. The filter kernel \( k(s) \) restricts the domain of integration and need not be the same for all points on the filtered slice. The kernel may depend on additional parameters such as data derived from the vector field.

Weiskopf [9] chose a texture-based approach for storing and processing the dense representation of the property field, the vector field, and some additional auxiliary data. A rough outline of the texture-based discretization 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 \( F \) within the resulting three dimensional property field.

Each particle has a lifetime of \( \tau \), and therefore each trajectory has a maximum length. As mentioned above, the property of a particle may change over its lifetime using, for example, a weight function \( w(t_{age}) \) where \( (t_{age}) \in [0, \tau] \). Newly introduced particles are described by field \( P(x, t) \) where a particle born at \( (x_0, t_0) \) is described by \( P(x_0, t_0) \). The contribution to the property field \( I \) at time \( t \) by a particle born at position \( x_0 \) at an earlier time \( t_0 \) (within the lifetime) is therefore:

\[
I(x_{path}(t; x_0, t_0); t) = w(t - t_0)P(x_0, t_0)
\] (32)

By adding contributions from all possible particle paths originating from different discrete times \( t_i \) between \( (t - \tau) \) and \( t \), one obtains the property field

\[
I(x, t) = \sum_{t_i} w(t - t_i)P(x_{path}(t_i; x, t))
\] (33)

\( P_i \) denotes a spatial slice of \( P \) at time \( i \). Being a texture, \( P_i \) can represent any distribution of released particles from white noise to frequency-filtered noise or some input with large-scale structures used for dye injection. 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. However, the limited lifetime of particles ensures that these differences do not grow without bound and therefore subsampling and filtering is usually not required.

Inflow regions, such as parts of the domain in which the backward integration of trajectories leaves the boundaries of the textures \( P_i \) are addressed by enlarging 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 clamping texture coordinates to the texture boundaries.

“The main features of this implementation of UFAC in a 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 \( w(t_{age}) \)” [9]

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.

9. Conclusion
This survey reviewed some of the techniques for flow visualization that advanced the practice beyond arrow plots and glyphs. Although arrow plots and glyphs can portray fluid vector field velocity, direction, acceleration, and can apply to three dimensional and time-dependent fields, the size of the area represented must be small since they use a considerable amount of spatial resolution. Particle tracing techniques offer an intuitive representation of transport along the flow. This survey did not address sparse representation techniques in which the spatial domain is not densely covered. Dense representations for particle-tracing methods typically are built on texture-based techniques which provide images of high spatial resolution. This survey attempted to address some of the more significant developments in this area. Spot noise and Line Integral Convolution (LIC) offered early and quite useful methods of particle tracing techniques. LIC in particular was the basis for other techniques that sought to address limitations and offered improved performance such as dye advection, Oriented LIC (not discussed in this survey), Unsteady Flow LIC (UFLIC), and Dynamic LIC (DLIC). The survey explored texture advection techniques in which a dense collection of particles in a texture are transported according to the motion of particles in a time-dependent unsteady vector field. This survey looked at two versions of the same ideas, Langrangian-Eulerian Advection (LEA) and Image-Based Flow Visualization (IBFV). They visualize unsteady flow by integrating particle positions and advecting the color of the particles based on a texture representation. In LEA, both noise and dye advection can be handled in the same framework. LEA can be extended to three dimensions. In IBFV, the noise texture is advected and a second texture is blended into the advected texture at each time step. IBFV cannot be extended to three dimensions [10]. Unsteady Flow Advection-Convolution offers the user control of spatial and temporal coherence by providing explicit and separate control of the patterns.

Although LEA achieved interactive frame rates, more development is needed in this area to support user exploration of three dimensional vector fields. The advances in GPU processing and multiple processors make interactive frame rates with LEA possible. GPU processing achieves a much higher processing speed, potentially two to three orders of magnitude higher than a comparable CPU-based implementation. This type of performance might make the difference between real-time visualizations allowing for effective user interaction and the non-interactive methods [11]. Occlusion of dense representations in three dimensions remains a challenge. Although not covered in this survey, feature-based visualization approaches have been developed. These methods seek to compute a more abstract representation that already contains the important properties in a condensed form and suppresses superfluous information. [10]

This survey of texture based methods for visualization of time-dependent vector fields helped the author discover the rich knowledge base on the subject. The author also discovered a branch field of fluid flow visualization of shock waves while reviewing Weiskopf et al. [10]. Yet to be explored by this author are shock-detection algorithms with edge-detection methods and visualization of flow separation and attachment.

I spent about 150 hours on this project. The sources that I used included the course wiki for the initial idea which started with exploring texture based methods for time-dependent visualization by Weiskopf [9]. The list of references provided a starting point for searching the ODU library.
10. References


