Visualizing Vector Fields Using Line Integral Convolution and Dye Advection

Han-Wei Shen[†]
MRJ
NASA Ames Research Center

Christopher R. Johnson
Department of Computer Science, University of Utah

Kwan-Liu Ma[†]
Institute for Computer Applications in Science and Engineering

Abstract

We present local and global techniques to visualize three-dimensional vector field data. Using the Line Integral Convolution (LIC) method to image the global vector field, our new algorithm allows the user to introduce colored "dye" into the vector field to highlight local flow features. A fast algorithm is proposed that quickly recomputes the dyed LIC images. In addition, we introduce volume rendering methods that can map the LIC texture on any contour surface and/or translucent region defined by additional scalar quantities, and can follow the advection of colored dye throughout the volume.

[†]This research was supported in part by the National Aeronautics and Space Administration under NASA contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

1 Introduction

Visualizing vector field data is challenging because it is intrinsically difficult to visually convey large amounts of three-dimensional directional information. In fluid flow experiments, external materials such as dye, hydrogen bubbles, or heat energy are injected into the flow. The advection of these external materials can 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. Numerical methods and three-dimensional computer graphics techniques have been used to produce graphical icons such as arrows, motion particles, stream lines, stream ribbons, and stream tubes that act as three-dimensional depth cues. While these techniques are effective in revealing the flow field's local features, the inherent two-dimensional display of the computer screen and its limited spatial resolution restrict the number of graphical icons that can be displayed at one time. In addition, interactive guidance that shows the user where to probe locally to obtain specific information is usually not available.

Additional techniques for vector field visualization include global imaging techniques. Crawfis and Max [1] [2] proposed direct volume rendering methods to create images of entire vector fields. Vector kernels and texture splats are used to construct three-dimensional scalar signals from the vector data. van Wijk [3] proposed a Spot Noise method using stretched ellipses to create two-dimensional textures that can be mapped onto parametric surfaces. Max *et al.*[4] further utilized the spot noise method to visualize three-dimensional velocity fields near contour surfaces. Cabral and Leedom [5] presented a Line Integral Convolution (LIC) method, which uses a one-dimensional low pass filter to convolve a white noise texture based on the directional information of the vector field. These methods can successfully illustrate the global behavior of vector fields; however, little or no user probing capability is provided, so specific information about the local behavior of the field is limited. Furthermore, some of these methods are difficult to apply to unstructured meshes.

In this paper, we present methods that integrate local and global visualization techniques to explore three-dimensional vector field data on regular grids. Using the Line Integral Convolution method as the underlying algorithm, we enable local probing by allowing the user to introduce "dyes" of various colors into the 2D/3D LIC flow field. The inserted dye propagates through the flow field, highlighting local flow features such as wavefronts, while the standard LIC texture still illustrates the global motion. We use three-dimensional direct volume rendering techniques that can map the LIC textures onto any contour surfaces or translucent region, and also can display the propagation of the dye through the volume.

We begin the paper by giving an overview of the LIC algorithm. Next, we describe the method of dye insertion. We propose a fast algorithm that can significantly reduce the time needed to recompute the LIC images upon insertion of new dye material. We then describe

the convolution kernel and how to control the advection distance of the dye. We also describe direct volume rendering methods that are used to render three-dimensional LIC data. We conclude the paper by presenting results of applying the techniques on a variety of large-scale scientific data sets.

2 Background

In this section, we give an overview of the LIC algorithm originally proposed by Cabral and Leedom [5]. Although the method can be applied to both two- and three-dimensional vector data, in this section we restrict our discussion to the two-dimensional case to illustrate the basic algorithm. We omit the description of some recent improvements to the LIC algorithm subsequently proposed by Forssell and Cohen [6] and by Stalling and Hege [7]. However, we note that these newer methods can be easily adapted to our new algorithm.

The LIC algorithm takes a vector field and a texture as inputs. The input texture is usually white noise data with the same resolution as the vector field. The output of the algorithm is a scalar field resulting from a local blurring of the input noise texture. The LIC algorithm carries out the local blurring by applying a one-dimensional low pass filter convolution throughout the input texture. The convolution kernel follows the direction of the streamlines originating from the corresponding grid points of the vector field in both positive and negative directions. As a result, the intensity values of the output scalar field are strongly related to the vector field's local flow direction. This convolution can be expressed as follow:

$$F_{out}(x,y) = \frac{\sum_{i=0}^{l} F_{in}(P_i)h_i + \sum_{i=0}^{l'} F_{in}(P_i')h_i'}{\sum_{i=0}^{l} h_i + \sum_{i=0}^{l'} h_i'} \tag{1}$$

$$h_i = \int_{s_i}^{s_i+\Delta s_i} \kappa(w)dw \tag{2}$$

Where

- $F_{out}(x,y)$ is the output pixel value at point (x,y).
- $F_{in}(P_i)$ is the input pixel value at point P_i.
- l and l' are the convolution distances along the positive and negative directions, respectively.
- P_i represents the ith cell the streamline steps in the positive direction, and P_i' represents a step in the negative direction.
- $P_0 = (x,y)$.

- h_i and h_i' are the weighting variables computed from the exact integral of the convolution kernel $\kappa(w)$.
- $\kappa(w)$ is the low pass filter used for the LIC.
- Δs_i is the arc length between the point s_i and s_{i+1} along the streamline.
- $s_0 = 0$.

The low pass filter used in [5] is a Hanning ripple function, which has a period of 2π. By shifting the phase of the filter function while performing the convolution, the algorithm can generate a sequence of LIC images to create a periodic motion effect.

3 Dye Injection

The intensity value of each LIC output cell can be represented as an average of input texture values along the streamline. This leads to the result that each output cell has an intensity correlated with cells along the streamline, but not with other cells. The use of the white noise input to the LIC algorithm assures that the boundaries of neighboring streamlines are not obscured. In addition, the LIC algorithm changes the phase of the convolution filter to "push" the noise texture along streamlines.

Through enhancing the individual streamlines and moving the noise texture, the LIC algorithm provides an excellent visual representation of the flow motion in the vector field. However, because of the nature of the LIC method, the correspondence between neighboring streamlines is difficult to observe. Sometimes, the user needs an observable correspondence in order to track local flow features, such as wavefronts. To resolve this discrepancy, we propose to use local flow visualization techniques, namely the introduction of foreign materials into the flow, and to observe the advection of these materials within the flow field. In the next section, we present a technique for injecting dye into the LIC field to highlight the flow field's local features.

3.1 Dye Smearing

The LIC algorithm can be used to create a "smearing" effect by convolving the filter with an input image. This allows the user to generate motion blur and additional artistic effects, as demonstrated in [5]. The dye insertion method utilizes LIC's natural "smearing" to simulate advection of dye within the flow field. We simulate the dye injection by assigning colors to isolated local regions in the input white noise texture. Cells whose streamlines pass through such regions receive color contributions from the dye. In addition, the phase shifting of the LIC algorithm can push the concentration of the dye along the streamlines. This creates the

effect of dye propagation. In the standard LIC algorithm, the convolution for each cell is performed in both positive and negative streamline directions to conserve symmetry. If we directly apply the standard LIC method to the input dyed texture, the upstream of the dyed area will be colored because the cells' positive streamlines in that area will pass through the dyed area and obtain the color contribution. However, to create a correct motion effect, the dye should smear only in the forward direction of the flow field. Therefore the dye should color only those cells in the downstream direction that correspond to cells whose negative streamlines pass through the dyed areas. To overcome this difficulty, initially a regular LIC image $F_{out}(x, y)$ for each animation step is computed using the white noise input. When the user injects the dye, we apply the LIC convolution using the dyed texture input to those cells that will be affected by the dye. The convolution is applied along these cells' backward streamline directions to ensure that the dye will only smear forwards. The results are then stored into $D_{out}(x, y)$. The final image of the LIC with dye advection can be obtained by using the formula:

$$Final_{out}(x, y) = D_{out}(x, y) \otimes F_{out}(x, y) \tag{3}$$

The operator $\otimes$ overwrites the standard LIC pixel values by the dyed values.

An important issue that must be addressed stems from the fact that every time the user injects a new dye, only a small portion of the cells are affected by the dye. In the next section, we propose a fast searching algorithm to rapidly locate those cells affected by the injection of new dye.

3.2 Fast Searching Algorithm

From our previous description, we know that only cells whose negative streamlines pass through the dyed region within a distance of the convolution length receive color contributions from the dye. We define a relation:

Definition 1 *For cells α and β, $\alpha \rightarrow \beta$ if and only if there is a backward streamline flowing through α and stepping into β*

Given a dyed cell α, the set of cells, $\Omega(\alpha)$, that are affected by this dye can then be expressed as:

Axiom 1 $\Omega(\alpha) = \{\rho | \rho \rightarrow \alpha, d(\rho, \alpha) \leq L\}$

where $d(\rho, \alpha)$ is the arc length of the streamline from point ρ to point α, and L is the convolution length.

Given a cell α, the goal is to quickly locate the set of cells $\Omega(\alpha)$ without searching through the entire field. A brute force method is to create a list for each cell. The list contains all the

cells that step backwards along streamlines into that cell. This information can be gathered while we first perform the standard LIC, since all the necessary streamlines are computed at that time. To decide upon the set $\Omega(\alpha)$, we can simply traverse through the list and retrieve the cells.

While the above method is simple, the memory requirement to perform it would be overwhelming. This is because each cell's list would contain many cells along the streamlines. To resolve this, we have designed a new algorithm. First, we define a new relation:

Definition 2 *For cells α and β, $\alpha \xrightarrow{*} \beta$ if $\alpha \rightarrow \beta$ or $\exists\{\rho_0, \rho_1, \cdots \rho_n\}$, such that $\alpha \rightarrow \rho_0, \rho_0 \rightarrow \rho_1, \cdots \rho_n \rightarrow \beta$.*

From definition 1 and definition 2, we know that:

Axiom 2 *if $\alpha \rightarrow \beta$ then $\alpha \xrightarrow{*} \beta$.*

Note that the above axiom does not necessarily hold in the reverse direction. Secondly, define a set ω as:

Definition 3 $\omega(\alpha) = \{\rho | \rho \xrightarrow{*} \alpha, d(\rho, \alpha) \leq L\}$

From axiom 2, we know that:

Axiom 3 $\omega(\alpha) \supseteq \Omega(\alpha)$.

Our fast searching algorithm can locate the cells in the set $\omega(\alpha)$ that are a superset of $\Omega(\alpha)$. In practice, the difference between the number of cells in the set $\Omega(\alpha)$ and the set $\omega(\alpha)$ is very small. To locate the cells in ω, we define a new relation:

Definition 4 *A cell $\alpha \xrightarrow{d} \beta$ if and only if $\alpha \rightarrow \beta$ and α is β's direct neighbor.*

We call this relation a *direct flow-back* relation. The *direct neighbors* of a cell are those cells that are directly adjacent to that cell in the physical space. For example, a cell has eight direct neighbors in a two-dimensional Cartesian grid, and has 26 direct neighbors in three-dimensional space.

The direct flow-back neighbors for each cell can be obtained and stored when we first use the standard LIC algorithm. Given a root cell α, the cell and its recursive direct flow-back neighbors actually constitute a directed graph. The nodes in this directed graph are then the members of the set ω. Therefore, the $\omega(\alpha)$ can be found by using:

Algorithm 1 *A member of $\omega(\alpha)$ can be found by using a* Breadth First Search *method starting from cell α and traversing through its direct flow-back neighbors recursively.*

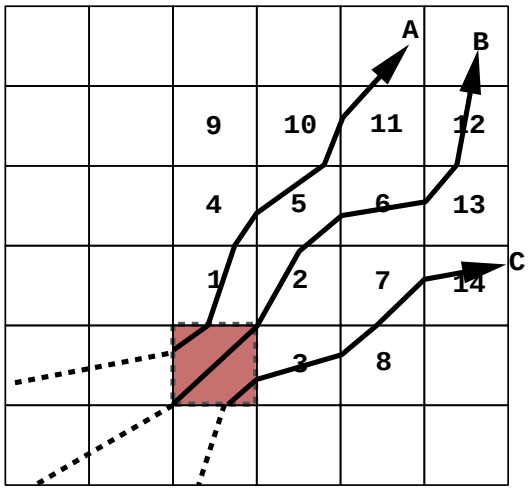

Figure 1: Streamlines and dye

Figure 1 shows an example of a flow field. The shaded cell is the original dyed cell. Figure 2 illustrates the resultant directed graph if we only consider the streamline A, B, and C.

Our new algorithm is considerably more memory efficient than the brute force method because only the direct flow-back neighbors need to be stored for each cell. The number of direct flow-back neighbors will usually be much lower than the actual number of the direct neighbors for a cell, with the exception of cells near critical points. We have found that the average number of direct flow-back neighbors for each cell in our experimental three-dimensional flows is approximately two or three.

3.3 Convolution Kernel

The Hanning windowed filter used in [5] can not be easily adopted by our algorithm to create natural dye advection effect. This is because that the Hanning ripple function is a periodic function with multiple peaks, and the Hanning window function attenuates the values of the incoming function values at two ends. Even though several parameters of these two functions could be modulated, it is still difficult to construct a good combination of the parameters such that the concentration of the dye can advect along the streamline smoothly. In our implementation, we use a simple box filter instead of the Hanning filter so that the appearance of the dye advection can be controlled more easily. The equation 2 becomes:

$$h_i = \int_{s_i}^{s_i+\Delta s_i} 1 dw = \Delta s_i \tag{4}$$

To add the ability to shift the filter box to create flow motion in the animation sequence, we revise the equation 1 to:

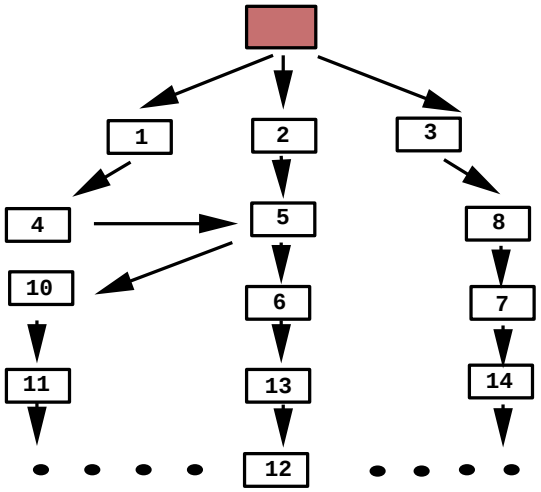

Figure 2: Flow-back directed graph

$$F_{out}(I,x,y) = \frac{\sum_{i=0}^{l} F_{in}(P_i)h_iT(I,s_i) + \sum_{i=0}^{l'} F_{in}(P_i')h_i'T(I,s_i')}{\sum_{i=0}^{l} h_iT(I,s_i) + \sum_{i=0}^{l'} h_i'T(I,s_i')} \quad (5)$$

Where

$$T(I,s_i) = \begin{cases} 1 & \text{if } s_i \in^* [\frac{I}{\Phi}L, \frac{I}{\Phi}L + \Delta L] \\ 0 & \text{otherwise} \end{cases}$$

- Φ is the number of LIC animation steps
- $I \in [0, \Phi - 1]$ is the current animation step
- ΔL is the length of the filter box

Figure 3 shows the phase shift of the box filter.

In equation 5, $T(I,s_i)$ is a predicate that determines if the current pixel is within the range of the filter box $[\frac{I}{\Phi}L, \frac{I}{\Phi}L + \Delta L]$. Note that if $\frac{I}{\Phi}L + \Delta L$ is greater than L, this value needs to be wrapped around so that it become $(\frac{I}{\Phi}L + \Delta L) \bmod L$. The new notation $\in^*$ that we use above is therefore based on this relationship.

It is known that the box filter could cause artifacts when the boxes reenter the interval. A solution to avoid this problem can be found in [7].

3.4 Advection Length

In the standard LIC algorithm, there is only a single convolution length defined for all cells in the field.[‡] This global convolution length determines the distance that the dye can travel

[‡] To simplify our explanation here, we omit the variation that the convolution length can be scaled based on the vector magnitudes.

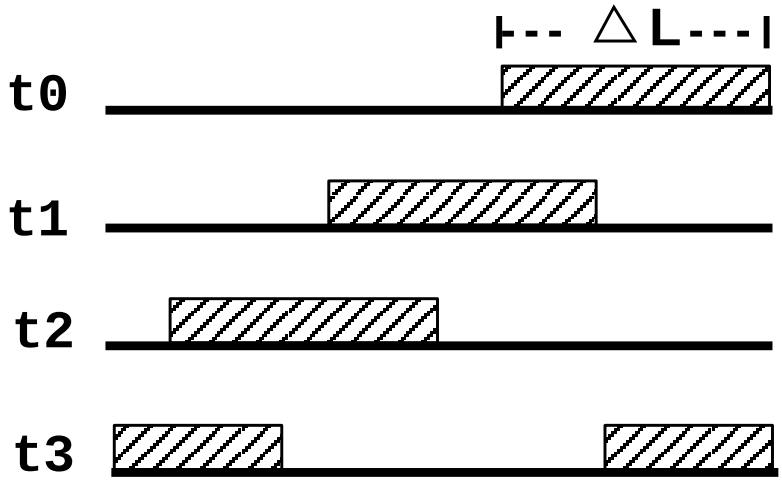

Figure 3: Phase shift of the box filter

and this distance is often desired to be long enough so that the local flow features can be clearly highlighted. However, extending the convolution length globally would decrease the contrast of the LIC texture and slow down the computation. In our algorithm, we allow the user to specify convolution lengths separately for the dye advection.

To use a longer convolution length for the dye advection, more LIC animation steps are needed to complete the animation. Assuming that the global convolution length is L, the number of animation steps for the standard LIC convolution is Φ and that the convolution length for the dye is Γ, then $\mathcal{I} = \Phi \times \frac{\Gamma}{L}$ is the number of animation steps required. Here we assume that Γ is always a multiple of L. We revise the LIC formula for the dyed advection: $D_{out}(I, x, y) =$

$$\frac{\sum_{i=0}^{-\Gamma} D_{in}(P_i) h_i T'(I, s_i)}{\sum_{i=0}^{-\Gamma} h_i T'(I, s_i)} \tag{6}$$

Where

$$T'(I, s_i) = \begin{cases} 1 & \text{if } s_i \in^* [\frac{I}{\mathcal{I}}\Gamma, \frac{I}{\mathcal{I}}\Gamma + \Delta L] \\ 0 & \text{otherwise} \end{cases}$$

- $D_{in}(P_i)$ is the input dyed texture
- $D_{out}(I, x, y)$ is the output value

for the dyed cell(x, y) at step I.

- $I \in [0, \mathcal{I} - 1]$ is the current animation step
- ΔL is the length of the filter box.

Note that for those cells that are not affected by the dye, only Φ steps of LIC output are computed. To combine these standard LIC outputs with the results from the LIC dye advection, we can use the formula:

$$Final_{out}(I, x, y) = D_{out}(I, x, y) \otimes F_{out}(I \bmod \Phi, x, y) \tag{7}$$

where the meaning of the operator $\otimes$ is explained in section 3.1.

4 3D LIC Rendering

While the output of the three-dimensional LIC algorithm is a scalar field, standard visualization techniques, such as isosurface extraction or volume rendering, cannot be directly applied to the output. This is because the values of the LIC volume consists primarily of noise signals that are used to construct the vector textures and do not themselves have any physical meaning. In this section, we present direct volume rendering methods that can render three-dimensional LIC textures on arbitrary surfaces or regions derived by additional scalar quantities, and also can display the propagation of the dye through the volume. In the sections that follow, we first introduce a *bi-variate volume rendering model* and then describe the methods used to image the dye propagation.

4.1 Bi-Variate Volume Rendering

To render the LIC texture onto local regions, our volume rendering process takes two volume data sets with separate transfer functions as the input. A scalar variable, such as the magnitude of velocity, or pressure, or of vorticity, is used as the *primary data*. The opacity map of the primary data, which is manipulated in the same way as in the standard volume rendering method, is used to define the transparency of each voxel in the volume. The LIC texture, which serves as the *secondary data*, has its own color map but uses the transparency defined by the primary data. To render these two input data sets, we define a *volume mixture*, which is similar to what is used in [8], to blend these two input data sets. The color of each voxel V, C_V, comes from the contributions of both data sets and can be expressed as:

$$C_V = \beta \times C_p + (1 - \beta) \times C_s, \tag{8}$$

where C_p is the color from the primary data at voxel V and C_s is the color from the secondary data. The $\beta \in [0, 1]$ is a weight between the primary and secondary data specified by the user. When β is 1, the volume mixture is equivalent to the scalar volume; when β is 0, the volume mixture is the same as the LIC volume. A gradual shifting of this parameter from 1 to 0 enables the user to track features from the scalar data at the beginning and visualize the counterpart of the vector data and vice versa. Our new rendering model allows the user to interactively change the weighting parameter and visualize the results.

Our bi-variate volume rendering method uses the standard front-to-back compositing method, where the final color at pixel p can be expressed by the following formula

$$C_p = \sum_{i=1}^{n} W(i) C_{V_i}, \tag{9}$$

where C_{V_i} is the color contribution at the i-th sample point V_i. $W(i)$ is the light attenuation factor, which is computed from the formula:

$$W(i) = \alpha(V_i)[1 - \sum_{j=0}^{i-1} W(j)]. \tag{10}$$

$\alpha(V_i)$ is the opacity at the point V_i and is defined by the primary data set's opacity map and its data value at that point. We can replace the C_{V_i} in equation 9 with the volume mixture in equation 8 and obtain:

$$C_p = \sum_{i=1}^{n} W(i) C_{V_i} =$$

$$\sum_{i=1}^{n} W(i)[\beta \times C_{V_i(p)} + (1-\beta) \times C_{V_i(s)}] =$$

$$\beta[\sum_{i=1}^{n} W(i) C_{V_i(p)}] + (1-\beta)[\sum_{i=1}^{n} W(i) C_{V_i(s)}]. \tag{11}$$

Here $C_{V_i(p)}$ is the color contribution at point V_i from the primary data set and $C_{V_i(s)}$ is the color contribution from the secondary data set. Equation 11 reveals a very interesting property: when we render the volume mixture, we can render each volume separately. The mixture image can be generated by a simple two-dimensional image blending.

Thus the bi-variate volume rendering model can be used to intermix the scalar and vector information. Moreover, the user can interactively adjust the volume blending parameter β to track important features from one variable and visualize the counterparts of the other variable.

4.2 Dye Rendering

It is helpful to display the LIC texture onto local regions such as isosurfaces and translucent areas. However, this could give the user the wrong impression that the flow is always along the surface or local regions. This problem can be remedied if we can visualize the dye propagation through the volume. In our method, we allow the user to specify opacity of the injected dye. The opacity values in the LIC dye input are convolved in the same way that we convolved the pixel values. As a result, the output volumes of the LIC computation consist of both color and opacity information.

To take the dye opacity into consideration when performing volume rendering, the bi-variate rendering model needs to be slightly modified. This is because the LIC volume, as a secondary data, also affects the volume's opacities. When rendering the LIC volume, we calculate the opacity value at each sample point as the sum of the opacities from the primary data and from the LIC volume. The color contribution of the LIC voxel at that sample point is then computed using this opacity. Following is pseudo code for rendering the LIC volume using ray casting.

```
for (each ray from the image plane) {
  for (each sample step along the ray) {
    opacity = primary opacity + dye opacity;
    accumulate the LIC color;
    accumulate the opacity;
  }
}
```

A simple but useful extension is that the color of dye can be determined by additional scalar variables. In this way both the flow direction and scalar distribution can be visualized at the same time.

5 Results and Discussion

We have implemented our algorithms using C++ and OpenGL. The software combines modules of Line Integral Convolution with Dye Injection and Bi-variate volume rendering. In our current software implementation, the user can inject dye by slicing through the three-dimensional LIC output and specifying the dye locations. Our fast search algorithm can rapidly recompute the LIC output based on the dye position and then feed the results back to the volume renderer module. We have found that our fast search algorithm can reduce the compute time, on average, by over 90% when compared to the standard LIC method for recomputing the animation sequence of the LIC data. The output of the LIC volume contains red, green, blue and opacity components for each voxel. For the volume rendering module, we define a C++ volume renderer class. The renderer instance of this class can be either a primary data renderer or a secondary data renderer. Through a software pointer between the primary data renderer and secondary data renderer, the secondary renderer can access the primary renderer's transfer function, volume data, and other auxiliary data structures to facilitate the bi-variate rendering model. A secondary renderer class instance is created for each LIC output in the animation sequence. These renderer objects can be dispatched to different processors in a multiprocessor environment to perform a coarse grain parallel volume rendering.

Figures 4-6 show several LIC images with dye propagation. Figure 4 shows three different colored dye propagating through a two dimensional vector field. Figure 5 is an animation sequence of dye advection in a $32 \times 30 \times 30$ vector field from a three-dimensional combustion simulation. Figure 6 is an animation sequence of dye advection in a $96 \times 96 \times 96$ tornado data set. The dye is colored based on the magnitude of the vector field.

6 Conclusion and Future Work

We have presented new algorithms to visualize three-dimensional vector fields using Line Integral Convolution methods. Our algorithm allows the user to insert dye into the flow field to enhance local flow features. Given a region with dye, our algorithm can rapidly locate the cells that are on the path of the dye propagation. Therefore, the LIC computation needs to be applied only on a small subset of the entire field. We proposed a bi-variate volume rendering method to display three-dimensional LIC textures on arbitrary contour surfaces or translucent regions that are derived from additional scalar quantities. The new rendering methods can also be used to render the dye propagation through the volume.

Future work includes providing more flexible control for modeling the behavior of the dye. This includes using different filter kernels to control the appearance of the dye. An additional useful feature would be to provide a natural and effective way for the user to probe the three-dimensional space and inject the dye. Finally, we plan to utilize the coherence between LIC volumes in an animation sequence to speed-up the volume rendering and allow for interactive data exploration.

Acknowledgments

This work was supported in part by awards from the NSF, NIH, and by ICASE under NASA contract NAS1-19480. We would like to thank C. Hansen, K. Coles, R. McDermott, and the reviewers for their helpful comments and suggestions. Furthermore, we appreciate access to facilities that are part of the NSF STC for Computer Graphics and Scientific Visualization.

References

[1] R. Crawfis and N. Max. Direct volume visualization of three-dimensional vector fields. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 55–60. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[2] R. Crawfis and N. Max. Texture splats for 3d scalar and vector field visualization. In *Proceedings of Visualization '93*, pages 261–265. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[3] J.J. van Wijk. Spot noise: Texture synthesis for data visualization. *Computer Graphics*, 25(4):309–318, 1991.

[4] N. Max, R. Crawfis, and C. Grant. Visualizing 3d velocity fields near contour surfaces. In *Proceedings of Visualization '94*, pages 248–255. IEEE Computer Society Press, Los Alamitos, CA, 1994.

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

[6] L.K. Forssell and S.D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transaction on Visualization and Computer Graphics*, 1(2):133–141, 1995.

[7] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH 95*, pages 249–256. ACM SIGGRAPH, 1995.

[8] R.A. Derbin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of SIGGRAPH 88*, pages 65–74. ACM SIGGRAPH, 1988.

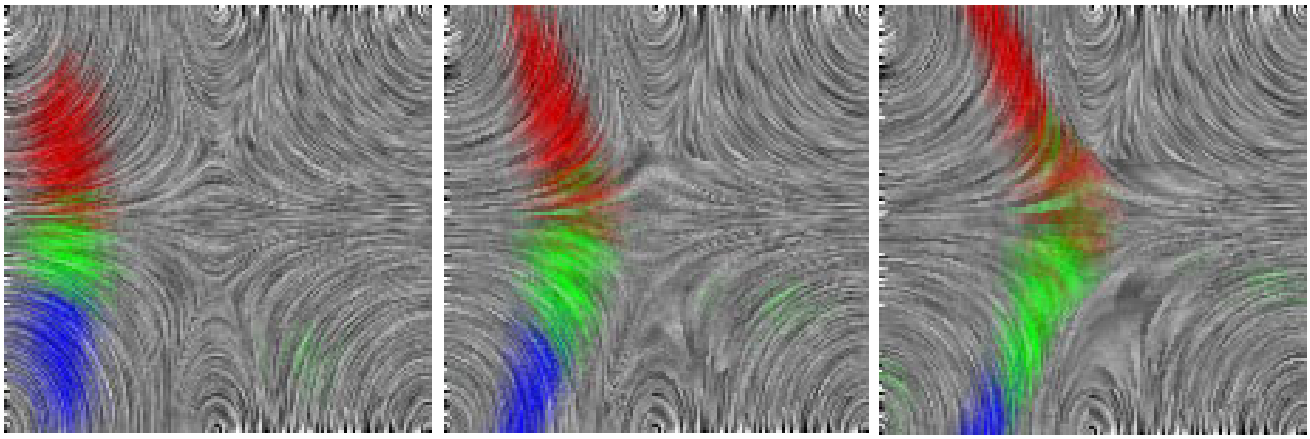

Figure 4: 2D LIC images with dye

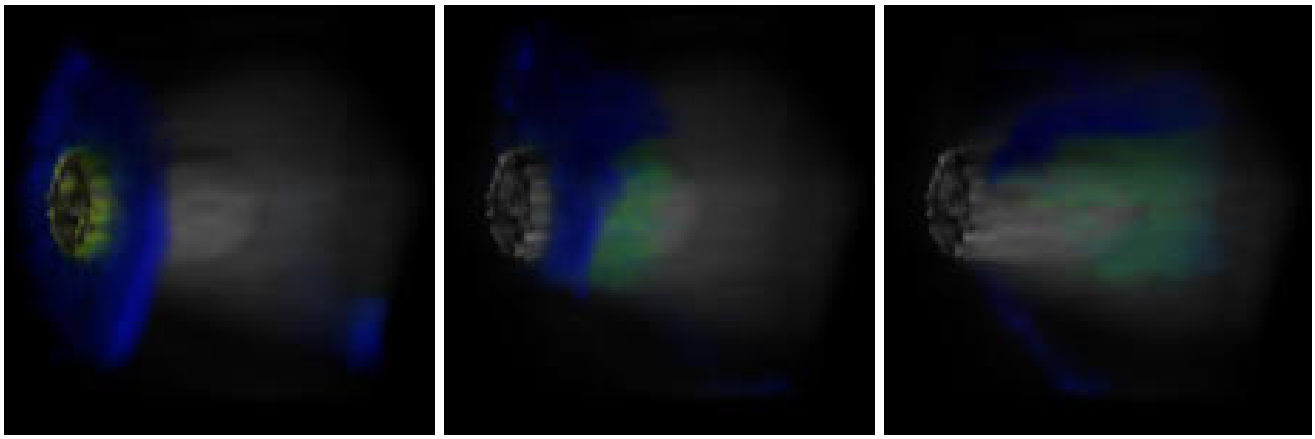

Figure 5: Dye advection in a 3D combustion simulation

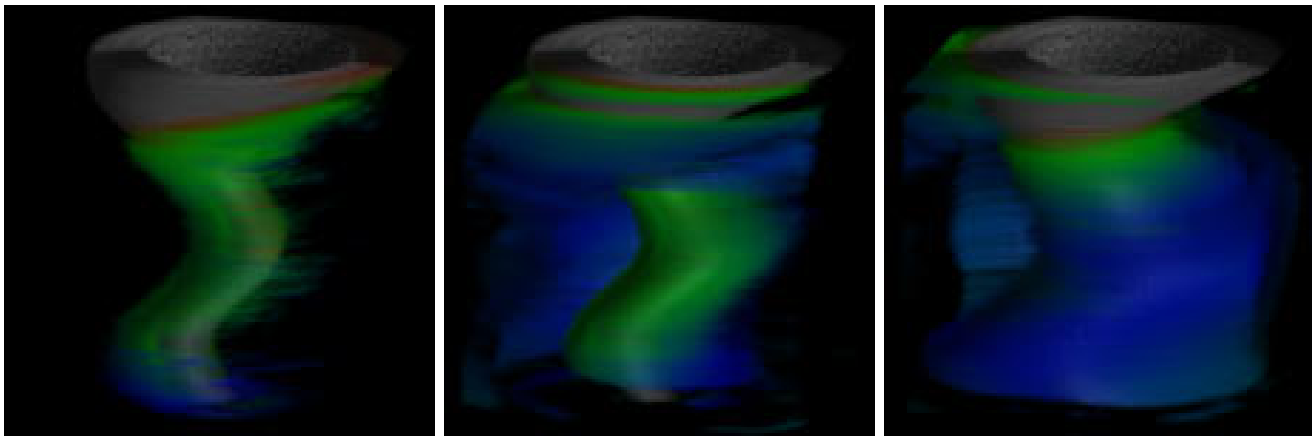

Figure 6: Dye advection in a 3D tornado simulation