Global and Local Vector Field Visualization Using Enhanced Line Integral Convolution

Han-Wei Shen† Christopher R. Johnson† Kwan-Liu Ma‡

† Department of Computer Science
University of Utah
Salt Lake City, UT

‡ ICASE
NASA Langley Research Center
Hampton, Virginia

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 a volume rendering method that can display the LIC texture on any contour surface and/or translucent region defined by additional scalar quantities. This new rendering model can be used not only to render the three dimensional LIC data, but also to explore data sets that have multiple scalar fields.

1 Introduction

Visualizing vector field data is challenging because no existing natural representation can 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. Analogues 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 avaliable.

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. We use the line integral convolution method as the underlying algorithm, but we add two very useful improvements. The first improvement permits 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. The second improvement uses the direct volume rendering method to map the LIC textures onto any contour surfaces or onto translucent regions derived from multiple scalar quantities. While the LIC algorithm can be directly applied to three-dimensional vector fields, there is no systematic post-processing method avaliable to render the resultant LIC data. We propose a rendering model that can intermix LIC images with other scalar data. This rendering model can also be used to explore the cause-and-effect relationships of multiple scalar fields within the same computation grid.

We begin the paper by giving an overview of the LIC algorithm. Next, we describe the method of dye insertion. We then propose a fast algorithm that can significantly reduce the time needed to recompute the LIC images upon insertion of new dye material. Then we describe the new multivariate direct volume rendering method and demonstrate how to use this method to render 3D LIC data. We conclude the paper by presenting results of the new local and global flow algorithms 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 will 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:

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

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

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 $\kappa(w)$.
- k(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, algorithm can generate a sequence of LIC images to create a periodic motion effect. In addition, the convolution length L, can also be scaled according to the magnitude of the vector velocity to create a variable speed effect for subsequent animation of the flow field.

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 regular images. This allows the user to generate motion blur and additional artistic effects, as demonstrated in [5]. The dye insertion method utilizes the 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. Assuming that the LIC's convolution length in both positive and negative directions is L, the period of the convolution kernel is P, and the phase shift at each step in a animation sequence is S, the dye smearing will have the following characteristics:

- The distance the dye travels at each step $= L \times \frac{S}{P}$
- The maximum distance the dye will travel $= L$

To correctly and efficiently model the dye propagation through the flow field, we must address two issues.

First, 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 cells upstream of the dyed area will be colored because their positive streamlines will pass through the dyed area and obtain the color contribution. However, for the method to be effective the dye should smear only in the forward direction of the flow field and therefore should color only those cells in the downstream direction, which correspond to cells whose negative streamlines pass through the dyed areas. To overcome this, we ensure that, for each cell, the forward convolution always takes the original white noise texture without dye as input, and the backward convolution takes the new dyed texture as input. In this way, no colored dye in the downstream direction will affect those cells in the upstream direction.

The second issue that must be addressed stems from the fact that every time the user injects a new dye, the LIC data must be recomputed. However, since only a small portion of the cells are affected by the dye, it would be inefficient to recompute the entire LIC data. In the next section, we propose a fast searching algorithm to prevent the need to recompute the entire LIC data by rapidly locating 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 going 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 backward direction. Secondly, define a set ω as:

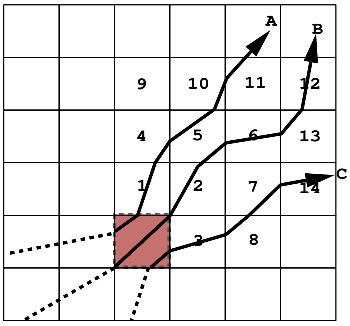

Figure 1: Streamlines and dye

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

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* Breadth First Search *method starting from cell* α *and traversing through its direct flow-back neighbors recursively.*

Figure 1 shows an example of a flow field. Cell α 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 3D flows is approximately two or three.

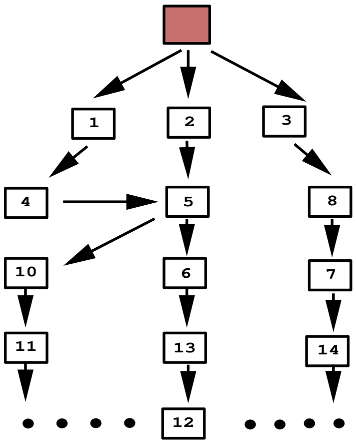

Figure 2: Flow-back directed graph

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 consist 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 a direct volume rendering model that can render three-dimensional LIC textures on arbitrary surfaces or regions. We can use other scalar variables, such as temperature, pressure, vorticity, or vector magnitude, to define local regions. The LIC texture and values of the scalar variable can then be displayed on those areas. To achieve this, we propose a *bi-variate* volume rendering model. This model can be used not only in rendering LIC texture, but also in exploring the cause-and-effect relationships between two scalar variables. In the sections that follow, we introduce the general concept of the bi-variate rendering model and then discuss how to use this model to render three-dimensional LIC data.

4.1 Bi-Variate Volume Rendering

There are many scientific applications whose simulations result in multi-variate data. In this section, we present a new method that can be used to explore the cause-and-effect relationships between two scalar variables by using direct volume rendering techniques.

Direct volume rendering using ray casting is an effective technique to visualize three-dimensional scalar fields. Transfer functions are used to map the data into visual attributes such as colors and opacities in order to enhance local features of the scalar field. In this section, we present a new volume rendering model, termed *bi-variate volume rendering*, that can be used to visualize scalar fields that have more than one variable. Our rendering model facilitates the exploration of cause-and-effect relationships between two scalar variables in two ways. First, we provide a simple and flexible approach that can be used to compare the value distributions of two variables at the same regions using a volume mixture method. Second, we allow the user the ability to gradually change the values of the volume mixture from one variable to the other and get the rendering results interactively. This smooth transition enables the user to track important features from one variable at the beginning and visualize the counterparts of the other variable.

The input to the rendering process contains two volume data sets with separate transfer functions. We use one of the data set's opacity maps to define the transparency of each voxel in the volume. This data set is called the *primary* data. The other data set is called the *secondary* data. 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 a blending of the two input data set. The color of each voxel V, C_V, comes from the contributions of both data and can be expressed as:

$$C_V = \beta \times C_p + (1-\beta) \times C_s, \tag{3}$$

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 primary volume; when β is 0, the volume mixture is the same as the secondary volume data. A gradual shifting of this parameter from 1 to 0 enables the user to track features from the primary data at the beginning and visualize the counterpart of the secondary data, and vice versa. Our new rendering model allows the user to interactively change the weighting parameter and obtain the result.

We describe our method as follows. Using the standard front-to-back compositing method, the final color at pixel p can be expressed by the following formula

$$C_p = \sum_{i=1}^{n} W(i) C_{V_i}, \tag{4}$$

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

$$W(i) = \alpha(V_i)[1 - \sum_{j=0}^{i-1} W(j)]. \tag{5}$$

$\alpha(V_i)$ 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 4 with the volume mixture in equation 3 and obtain:

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

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

$$\beta[\sum_{i=1}^{n} W(i) C_{V_i(p)}] + (1-\beta)[\sum_{i=1}^{n} W(i) C_{V_i(s)}]. \tag{6}$$

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 6 reveals a very interesting fact: 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. This facilitates the changing of the weighting parameter β and allows for rapid rendering.

The bi-variate volume rendering model can be used to intermix two scalar variables into a single image. The primary data set is used to enhance local regions with an appropriate opacity map. Values from either of the variables can then be color-coded onto those regions. Moreover, the user can interactively adjust the volume blending parameters to track important features from one variable and visualize the counterparts of the other variable.

4.2 LIC Rendering

Using the bi-variate volume rendering method, we can render the LIC texture onto local regions. A scalar variable, such as the magnitude of velocity, of pressure, or of vorticity can be used as the primary data.

The opacity map of this primary data is manipulated in the same way as in the standard volume rendering method to extract contour surfaces or translucent regions. The LIC texture, which serves as the secondary data, can then be mapped onto these local areas using the rendering method described previously. The user can change the weighting parameter β to choose to visualize the scalar or vector information on the local regions. When the β is appropriately chosen between $(0, 1)$, both the colors of the scalar variable and of the motion of the LIC texture can be seen; therefore, the scalar and vector data can be intermixed. To create the effect of periodic motion, we generate multiple LIC output files at a time, using the phase shift method described in [5]. These multiple LIC data sets all serve as the secondary data and are rendered and played back in order.

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 save an average of over 90% of compute time compared to the standard LIC method when recomputing the animation sequence of the LIC data.

The output of the LIC volume contains red, green, and blue components for each voxel. For the volume rendering module, we define a C++ volume renderer class. The renderer under this class can be either a primary data renderer or a secondary data renderer. Each renderer has its own volume data set. 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. Our software provides the capability of changing the transfer function and quickly updating the rendering results using the *Ray Cache* data structure [8]. This facilitates the users ability to define local areas and visualize the vector texture on those regions. 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.

Color plates 1-3 show several LIC images. Color plate 1(a) shows a standard LIC image for a two-dimensional 480×480 vector field. Plate 1(b) shows an image of the same vector field injecting three different colored dyes. Plate 1(c) shows a volume rendered LIC image for data from a combustion simulation. The data size is $32 \times 30 \times 30$ and the image size is 480×480. The contour surfaces are defined using the velocity magnitude as the primary data. Note that because the resolution of the vector data is relatively low compared to the image size, the LIC texture does not appear clear in the resultant image. By dropping colored dye into the field and performing a phase shift animation in the LIC computation, we can image the flow so that the directional information is significantly clearer. Plate 1(d) shows a frame in the animation sequence. Color plate 2 illustrates the bi-variate volume rendering method. The data consist of scalar fields corresponding to the stream function and vorticity on a $64 \times 64 \times 64$ regular grid. We used the stream function as the primary data to define the contour surface structure. Plate 2(a) shows a volume rendered image of the stream function, and plate 2(d) shows the vorticity image on stream function defined surfaces. Plate 2(b) and 2(c) illustrate the volume mixture of stream function and vorticity. Color plate 3 shows volume rendered images of a tornado data set with $96 \times 96 \times 96$ resolution. The image size is 480×480. Plate 3(a) illustrates contour surfaces defined by the vector magnitude, and plate 3(b) shows a translucent area and a more opaque surface inside. LIC textures are mapped onto those user defined regions using our bi-variate volume rendering method. Plate 3(c) and 3(d) show two frames in an animation sequence from the results of LIC with dye injection.

6 Conclusion and Future Work

We have presented new algorithms to visualize three-dimensional vector fields using Line Integral Convolution methods. The contribution of this paper can be divided into two parts. First, 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. The second contribution of this paper shows that three-dimensional LIC texture can be displayed on arbitrary contour surfaces or translucent regions that are derived from multiple scalar quantities. We propose a bi-variate volume rendering method to achieve this effect. This new rendering method can also be used to explore the cause-and-effect relationships between two scalar variables by intermixing their quantities into a volume mixture. The user can change the weighting parameter of the volume mixture and quickly obtain the rendered result.

Future work includes providing more flexible control for modeling the behavior of the dye. For example, currently the life span of the dye is dependent on the convolution length of the LIC kernel. However, it is necessary for the user to control the life span without affecting the convolution length globally. Another extension of this research will be to study the effect of using different filter kernels to control the appearance of the dye. An additional feature that would make the software more useful will provide a natural and effective way for the user to probe the three-dimensional space and inject the dye. Slicing through LIC data to specify the dye position is simple and effective; however, direct user input using point and click methods would be even more desirable. 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 exploring.

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, and R. McDermott 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] K.-L. Ma, M.F. Cohen, and J.S. Painter. Volume seeds: A volume exploration technique. *The Journal of Visualization and Computer Animation*, 2:135–140, 1991.