USING LINE-INTEGRAL CONVOLUTION TO VISUALIZE DENSE VECTOR FIELDS

Han-Wei Shen

Department Editor:
Steve Bryson
bryson@nas.nasa.gov

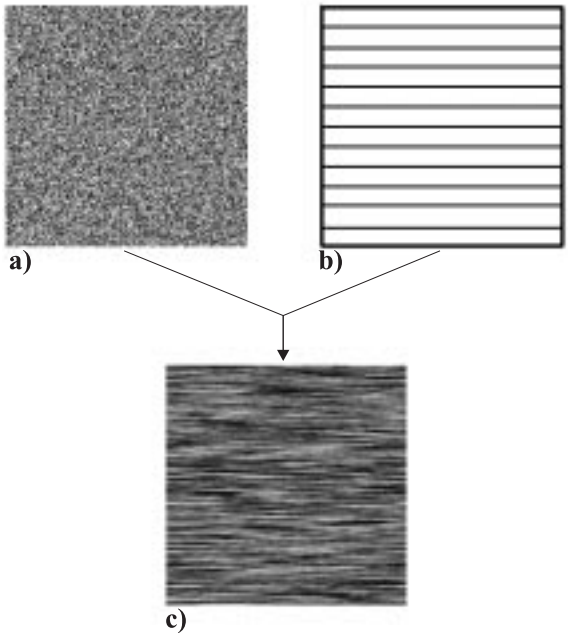

Figure 1. Demonstration of one-dimensional image convolution. Image (a) with random pixels, when convoluted with set of horizontal lines (b), yields horizontal texture pattern (c).

As high-performance computers containing large amounts of memory and disk space become more accessible, demands for more effective visualization methods that can analyze large-scale numerical data also continue to grow. Among many problems that are faced by visualization researchers, how to effectively visualize numerical vector field data is one of the most challenging research subjects. In 1993, Cabral and Leedom[1] proposed an innovative approach called line-integral convolution (LIC), which can produce realistic visualizations of flow direction everywhere in a vector-field. This technique has drawn a lot of attention from researchers as well as practical users in the past several years. Generally, people are impressed by the results of using LIC to visualize vector-field data. In this article, I briefly review this new visualization technique, along with some recent extensions, in the hope that you will find it useful for your own applications.

Visualizing dense vector-field data is a difficult problem. This is because the graphical icons that can naturally represent a vector tend to use up too much of the spatial resolution of the display. Suppose we were to draw a line segment representing a vector's direction and make the length of the segment proportional to the vector magnitude. Unlike displaying a scalar quantity, in which we can just paint a color to a single pixel to illustrate its value, drawing a line segment inevitably requires more screen space. As a result, the display window quickly gets cluttered as the size of vector data increases.

Another way to visualize the vector field is to specify seed locations and then compute streamlines, curves that are tangent everywhere to the flow direction in the field. This is one of the most popular visualization techniques in computational fluid dynamics. The disadvantage of using streamlines, however, is that in order not to miss any subtle features in the data, you have to know where to place those seed points. For large-scale simulations with complex geometric models, knowing for sure where the interesting features are is nontrivial. You probably will need to place a lot of streamlines, which again clutter the display, or you can probe around by trial-and-error, but you never know if you have missed anything.

To solve the problems, visualization researchers have been working on "global techniques," trying to image the flow direction everywhere in the field into a single picture. Such techniques attempt to represent the vector at every grid point by as little as one pixel display space and to pull out interesting features automatically. In this way, large data sets can be effectively visualized, and the amount of user interaction involved becomes minimal. In the following, I review the LIC algorithm and some of its extensions to show you how the technique realizes these goals.

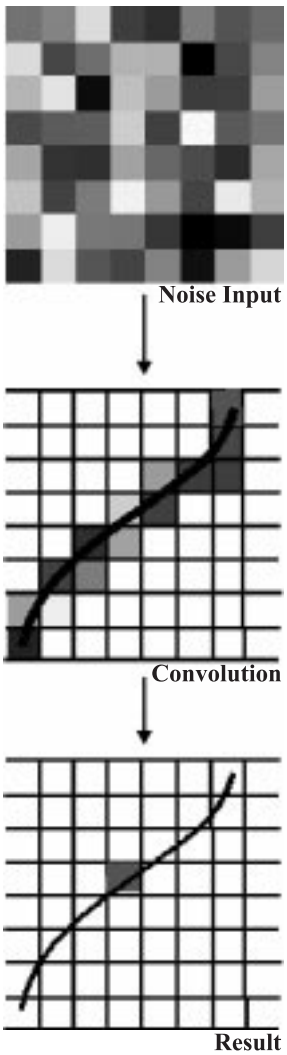

Figure 2. Process of line-integral convolution.

Convolution algorithm

Image convolution is a common technique in image processing. Given an input

Han-Wei Shen is a research scientist with MRJ Inc. at NASA Ames Research Center, Moffett Field, CA 94034. E-mail: hwshen@nas.nasa.gov

 S0894-1866(97)00805-5

image, each pixel value in the output image is computed by the weighted averaging of a small region of pixel values from the input. The weighting variable used to multiply the input pixel value is defined by a function called the convolution kernel. With reference to the dimension of the small region covered by the convolution kernel for each pixel, we can classify the operation as one-dimensional, two-dimensional, or three-dimensional convolution.

People often use "convolution" to mean smoothing an image or computing derivatives. However, image convolution can also be used to synthesize textures based on predefined patterns. For example, Fig. 1(a) is an image with random pixels. Figure 1(b) shows horizontal lines in a plane of the same size. Suppose we want to create a new image that has the horizontal texture pattern as shown in Fig. 1(b). To achieve this, we apply a one-dimensional convolution (the convolution region for each pixel will be a line) to the random image. For each pixel, the convolution is computed by averaging the values of the pixel itself, and its left and right five neighboring pixels along the horizontal direction. This average is then used as the pixel's output value. Once the process is finished, we get Fig. 1(c). As you can see, it shows the horizontal texture pattern that we want.

One-dimensional image convolution can be used to create textures of arbitrarily oriented lines. For vector-field visualization, we can synthesize images based on the flowline directions.

The original LIC algorithm proposed by Cabral and Leedom takes as inputs a vector field on uniform Cartesian grids and an image. The vector field can be two- or three-dimensional, and the image contains white noise with the same resolution as the vector field. To visualize the vector data, the LIC algorithm uses a one-dimensional low-pass filter function based on the flow-line directions in the vector field as the kernel to perform convolutions on the input noise image. The convolution region for each pixel is defined by streamlines originating from that pixel in both its positive and negative directions. These streamlines can be computed using numerical-integration schemes such as second- or fourth-order Runge-Kutta methods. The weighting variables used to multiply the input pixel values along the streamlines are computed by the exact integral of the convolution kernel function κ at every numerical-integration step of the streamline, which can be written as

$$h_i = \int_{s_i}^{s_i+\Delta s_i} \kappa(\omega)\, d\omega$$

where Δs_i is the ith step size (distance) of the numerical integration and s_i is the distance that the streamline has traveled after i steps:

$$s_0 = 0$$

$$s_i = s_{i-1} + \Delta s_{i-1}$$

Using the exact integration result h_i as the weighting variable, we can compute the convolution result at each pixel as:

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

where $F_{out}(x,y)$ is the output value at pixel (x,y), $F_{in}(P_i)$ is the input pixel value from the noise image at point P_i, P_i is the position of the ith streamline integration in the positive direction, P_i' represents the corresponding step in the negative direction, $P_0 = (x, y)$, l and l' are the convolution distances along the positive and negative directions respectively, and h_i and h_i' are the weighting variables as described above. Figure 2 illustrates the convolution process for a pixel.

In the above convolution, the user needs to specify a single parameter, the convolution length L, which is equal to the variables l and l' that are used in the above formula. When L is made larger, the convolution result becomes smoother, and the streamlines appear to be more continuous.

The convolution kernel is a low-pass filter function. Cabral and Leedom use a periodic low-pass filter, called the Hanning windowed function, which can be expressed as:

$$\kappa(\omega) = \frac{1+\cos(c\omega)}{2} \times \frac{1+\cos(d\omega+\beta)}{2}$$

$$= \frac{1}{4}(1+\cos(c\omega)+\cos(d\omega+\beta)+\cos(c\omega)\cos(d\omega+\beta))$$

where c and d are constants and β is the phase shift given in radians. The period of the function's phase is 2π.

Remember that we need the exact integration of the kernel function in the convolution. It is:

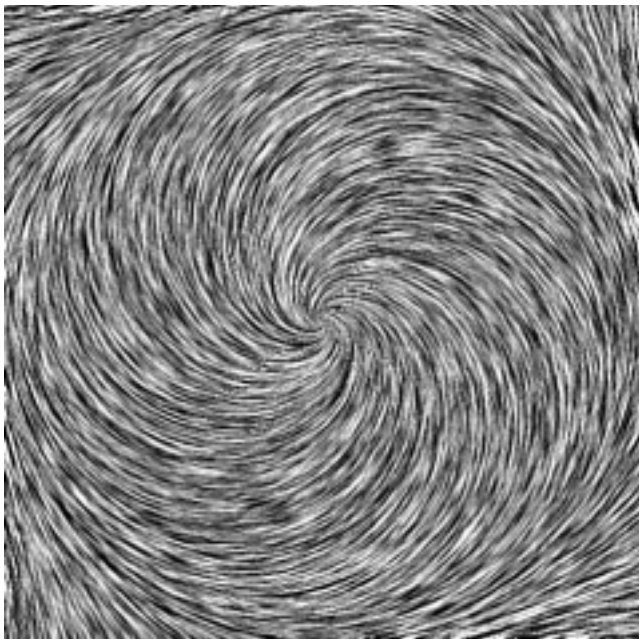

Figure 3. Visualization of a two-dimensional vector field using LIC.

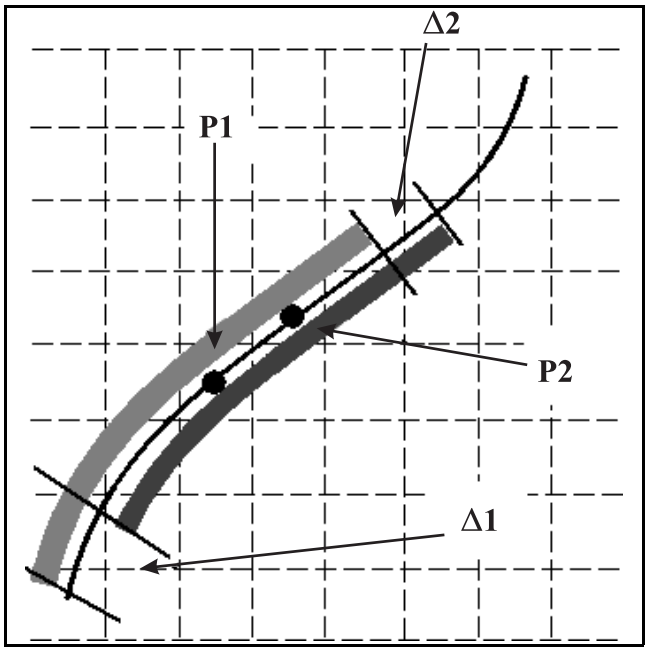

Figure 4. Convolution ranges of pixels P1 and P2.

$$\int_a^b \kappa(\omega)\, d\omega = \frac{1}{4} \times \left(\begin{array}{l} b - a + \dfrac{\sin(bc) - \sin(ac)}{c} \\ + \dfrac{\sin(bd + \beta) - \sin(ad + \beta)}{d} \\ + \dfrac{\sin(b(c-d) - \beta) - \sin(a(c-d) - \beta)}{2(c-d)} \\ + \dfrac{\sin(b(c+d) + \beta) - \sin(a(c+d) + \beta)}{2(c+d)} \end{array} \right)$$

Figure 3 shows an example of an LIC image.

A feature of LIC is its ability to animate the flow. In the above formula, the value of β represents the function's phase. The shape of this convolution kernel function will be shifted forward if its phase is stepped from 0 to 2π. Using different phases of the kernel function in the convolution, we can generate a sequence of images, in which the noisy "blobs" will appear to move forward, hence creating a realistic flow effect.

In addition to visualizing vector fields, LIC can be used to create interesting special effects.[1] The idea is to use a regular image and a vector field associated with that image as the inputs to the LIC. By computing the LIC based on the inputs, you can smear the colors in the image along the flow directions.

I have briefly reviewed the LIC algorithm. In the following, we look at some interesting extensions of LIC that have been developed.

Curvilinear grid data

Line-integral convolution is primarily for data on regular Cartesian grids. However, a majority of flow simulations generate data on structured curvilinear grids. Lisa Forssell extended LIC to work on such grids.[2]

For a curvilinear-gridded field, the location of any point in the domain can be represented by a physical coordinate system (in physical space) or by a computational coordinate system (in computational space). In computational space, data points are organized in a regular Cartesian grid, and cell elements are uniformly sized in all dimensions. In physical space, meshes conform to model geometries, and the spaces between grid points are nonuniform. Forssell proposed computing the LIC first in computational space, since data in this space have the Cartesian grid structure that LIC requires. Once the convolution is completed, the results can be transformed back to physical space using a standard inverse mapping. In Forssell's approach, if the vector data are defined in physical space, you need to transform them to computational space before the LIC computation can proceed. This can be done by multiplying the vectors by inverse Jacobian matrices.

In the LIC convolution, we need to specify a convolution length that a streamline, which defines the convolution path, can travel. This convolution length is used globally so that the streamline generated from each pixel travels the same distance in space. Since we compute the LIC in computational space, when we transform the result back to physical space for display, a given fixed convolution length actually corresponds to different distances for regions with different grid densities. This will cause a problem for us.

Why is this a problem? Remember that animations of flow motion can be generated by shifting the phase of the convolution kernel. In the animation, the flow speed appears to be affected by the amount of phase that is shifted at each animation step and convolution kernel length. The combination of these two factors decides the distance that a "ripple" can move in each frame. If the convolution kernel length is L and the convolution kernel phase is shifted by $1/n$ at each animation frame, then the distance that a "ripple" can move at each step is approximately $L \times 1/n$. When we transform the LIC result from computational space back to physical space, the convolution length L becomes shorter in regions that have higher grid densities. In the formula $L \times 1/n$, because L is now smaller, the flow speed will appear to be slower. Of course this is wrong, because we know that the flow velocity has nothing to do with the grid density. To amend this problem, Forssell proposed a solution that stretches the convolution kernel length in computational space for pixels that are in regions with higher grid density so that its corresponding length in physical space becomes the same everywhere.[2] To do this, the convolution kernel length at each pixel p is computed by

$$l(p) = a + b \times r(p) ,$$

where a is the fixed minimum kernel length for everyone, b is the maximum of extra convolution length, and $r(p)$ is the

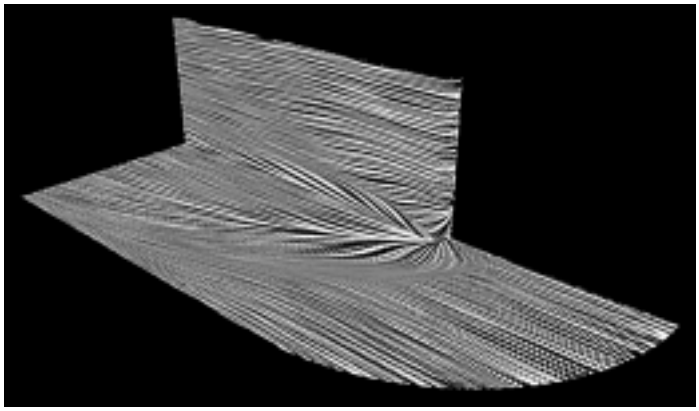

Figure 5. LIC image of a curvilinear-gridded blunt-fin data set.

grid density at cell p. Details can be found in Forssell's paper,[2] which includes a formula to compute the grid density. Figure 5 is an LIC image of curvilinear gridded blunt-fin data.

Accelerating the computation

Although LIC is effective at visualizing vector fields, unfortunately it is not fast enough to perform real-time data exploration. Stalling and Hege have proposed an extension that speeds up the process.[3] Their work is based on two key observations. First, a streamline starting from any point in the domain actually passes through many pixels. These pixels can be made to share this streamline when computing the convolution, so as to avoid redundant numerical integrations (numerical integration is expensive). The second observation is that adjacent pixels in the same streamline use similar pixel sets for the convolution. Therefore, the LIC value computed for one pixel can be reused by its neighbors with small modifications to accelerate the convolutions. By reducing the number of streamlines computed and reusing portions of the computation, Stalling and Hege achieve significant savings.

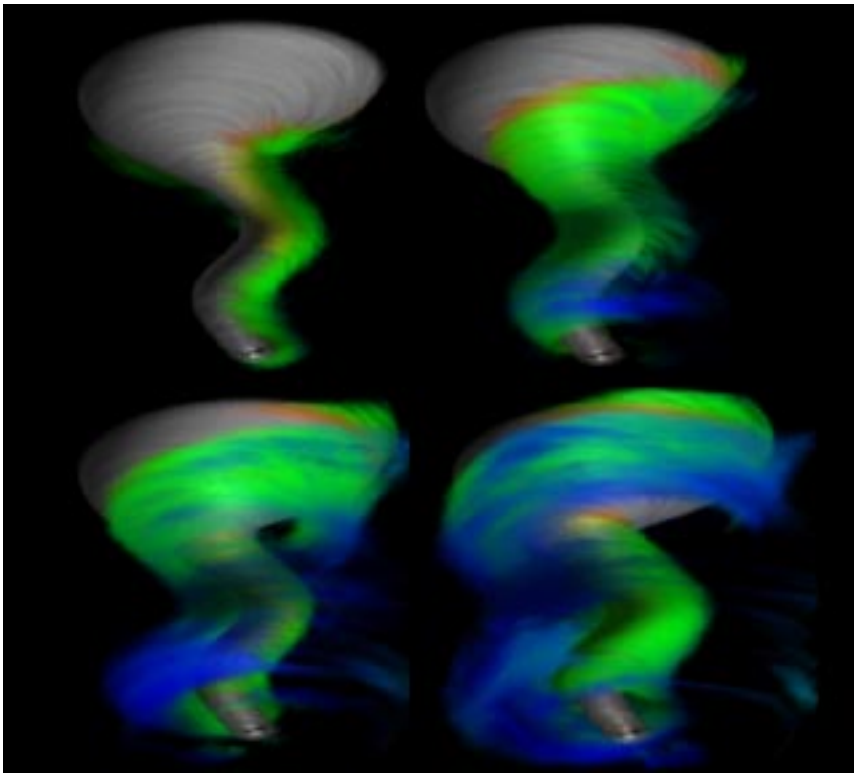

Figure 6. Snapshots from an animation of a three-dimensional tornado simulation visualized by LIC with dye advection.

Figure 4 illustrates the convolution ranges for pixels P1 and P2. The wide overlap area in the middle is used by both pixels. Assuming that the convolution output for P1 is $I(\mathrm{P1})$, then we can compute the convolution value for P2 as:

$$I(\mathrm{P2}) = I(\mathrm{P1}) - I(\Delta 1) + I(\Delta 2)$$

where $I(\Delta 1)$ and $I(\Delta 2)$ are the convolution values for areas $\Delta 1$ and $\Delta 2$. To make the above equality true, Stalling and Hege use a box filter, which is a constant function κ. With such a filter, the convolution result for pixel x_0 can be expressed as

$$I(x_0) = k \sum_{i=-L}^{L} T(x_i) \qquad k = \frac{1}{(2L+1)}$$

where x_i is the pixel at the ith step of the streamline integration, $T(x_i)$ is the input image (texture) value at x_i, and L is the convolution length. Once $I(x_m)$ is known ($m = 0$ in this case), we can compute $I(x_{m+1})$ and $I(x_{m-1})$ as:

$$I(x_{m+1}) = I(x_m) + k\,[T(x_{m+1+L}) - T(x_{m-L})]$$

$$I(x_{m-1}) = I(x_m) + k\,[T(x_{m-1-L}) - T(x_{m+L})]$$

The results for $I(x_{m+1})$ and $I(x_{m-1})$ are stored into pixel x_{m+1} and x_{m-1} respectively. The convolution then continues one step further in both streamline directions using the above formula.

With this convolution method, by computing a single streamline, we can "hit" many pixels along the line and quickly compute their LIC values. To reduce the total number of streamlines computed in the field, Stalling and Hege recommend starting the convolution from random pixels and keeping track of how many "hits" each pixel has received from streamlines originated from elsewhere. A new streamline is then initiated from a given pixel only if its number of "hits" is lower than some threshold value.

Stalling and Hege also propose other improvements on LIC such as relaxing the size requirements on input vector fields, input texture images, and output result images. In their paper they also describe methods that can produce smoother animation sequences.

Dye advection

LIC is very effective in visualizing flow directions in vector fields. However, sometimes we are interested in observing the correspondence between neighboring streamlines such as the propagation of wavefronts. Unfortunately, the convolution result given by LIC does not highlight such features clearly. To amend this problem, Shen (myself), Johnson, and Ma propose introducing dye advection into LIC computation.[4] The special effect known as "smearing" mentioned previously can readily be used to simulate the dye advection. Initially, we apply the regular LIC method to compute the image frames as usual. To inject dye into the field, we replace the noise values of the input pixels that serve as seed points with colors of the inserted dye, then recompute the LIC convolution. Due to the nature of LIC, these colors are smeared away by the flows.

Although the LIC convolution is computed globally, the dye advection will affect only local regions, that is, those pixels in the way of dye propagation. It would be inefficient to recompute the LIC of the whole field so as to represent only a small amount of dye injection. To avoid such inefficiency, we note that for a given dyed area, only the pixels with backward streamlines passing through this area will be af-

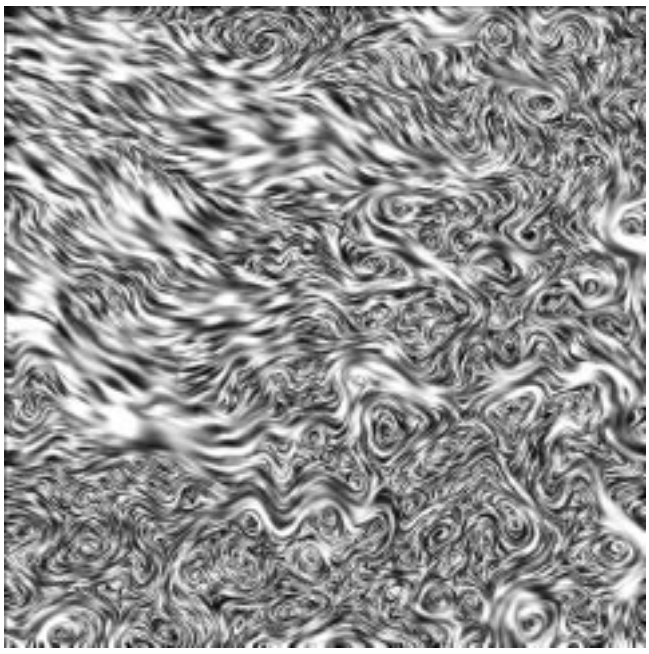

Figure 7. Image created by multifrequency LIC. Regions with different velocities have different texture densities.

fected by the dye. A fast search algorithm that creates a *direct flowback neighbor* list for each pixel and uses a broad first-search algorithm rapidly locates the dyed regions. LIC is then recomputed only in those regions.

In our paper we also demonstrate a volume-rendering model to display three-dimensional (3D) LIC results. To extend to three dimensions the original LIC algorithm proposed by Cabral and Leedom, a 3D noise volume is used instead of a noise image as the input texture. The streamline convolution is then performed in 3D space, and the LIC output is a 3D scalar volume. Direct volume-rendering techniques can be used, but with some modifications, to render this volume. Normally in volume rendering, the transfer function that specifies regions of interest is defined upon the scalar values of the volume itself. For volumes with LIC output, the transfer function must be defined by another scalar variable such as pressure, temperature, or magnitude of velocity. This is because the scalar values of LIC output are basically smoothed noise data and therefore cannot be used to define regions with meaningful physical characteristics. My colleagues and I propose a bivariate volume-rendering model, which can accept two scalar data sets, one as the *primary data* and the other as the *secondary data*. The primary data set is the scalar quantity used to define local regions, and the secondary data set is the LIC output that shows directional patterns on those regions. Figure 6 shows snapshots from an animation sequence of LIC with dye advection.

Multifrequency noise

Another extension of LIC worth mentioning is the multifrequency LIC method of Kiu and Banks.[5] Although LIC is excellent for illustrating the flow direction in a vector field, the magnitude of the velocity is usually difficult to discern in LIC images. To amend this problem, Kiu and Banks propose a method that varies the spatial frequency of the input noise texture based on the velocity magnitude in the field. Starting with the high-frequency noise input, they apply a Gaussian filter with different width w to the high-frequency noise image so as to create masks of different spatial frequencies. Because the Gaussian filter width w is inversely proportional to the velocity magnitude, larger blobs are created for regions with higher velocities.

To compute the LIC, masks with different noise frequencies representing regions with different velocities are assembled as the input noise texture. This multifrequency noise texture is then used by the regular LIC algorithm. Figure 7 is a result created by this technique.

Significant new developments

Line-integral convolution is a powerful method for imaging large-scale vector-field data. In contrast with interactive visualization, which requires a lot of user probing, LIC reveals important data features by displaying a dense array of information everywhere in the field. Recent developments have made LIC even more powerful. In addition to the techniques that I have reviewed here, other significant LIC techniques, which I do not have enough space to describe in detail, are Okada and Kao's postfiltering techniques for sharpening LIC output and highlighting flow features such as flow separations and reattachments (published in *SPIE Electronic Image '97*); Stalling, Zockler, and Hege's parallel line-integral convolution (published in *Proceedings of the First Eurographics Workshop on Parallel Graphics and Visualization*); Battke, Stalling, and Hege's fast line-integral convolution for arbitrary surfaces in 3D (published in *Visualization and Mathematics 1997* conference); and Mao *et al.*'s line-integral convolution for arbitrary 3D surfaces using solid texturing (published in *Eurographics Workshop on Visualization in SuperComputing 1997*). In addition, I and my colleague David Kao at NASA Ames will have a paper in this year's IEEE Visualization '97 conference about a new convolution algorithm for visualizing unsteady flow data. Up-to-date information on LIC can be found in the proceedings of the August 1997 SIGGRAPH conference, which had a tutorial on the subject (see http://www.icase.edu/~kma/lic_course97.html).

References

1. B. Cabral and C. Leedom, "Imaging vector fields using line integral convolution," in *Proceedings of SIGGRAPH 93* (ACM SIGGRAPH, New York, 1993), pp. 263–270.
2. L. K. Forssell and S. D. Cohen, "Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows," IEEE Transactions on Visualization and Computer Graphics **1**(2), 133–141 (1995).
3. D. Stalling and H.-C. Hege, "Fast and resolution independent Line Integral Convolution," in *Proceedings of SIGGRAPH 95*, (ACM SIGGRAPH, New York, 1995), pp. 249–256.
4. H.-W. Shen, C. R. Johnson, and K.-L. Ma, "Visualizing Vector Fields Using Line Integral Convolution and Dye Advection," in *Proceedings of 1996 Symposium on Volume Visualization* (IEEE Computer Society Press, Los Alamitos, CA, 1996), pp. 63–70.
5. M.-H. Kiu and D. Banks, "Multi-Frequency Noise for LIC," in *Proceedings of Visualization '96* (IEEE Computer Society Press, Los Alamitos, CA, 1996), pp. 121–126.