Visualizing Flow Over Curvilinear Grid Surfaces Using Line Integral Convolution

Lisa K. Forssell
Computer Sciences Corporation, NASA Ames Research Center
and Stanford University
lisaf@cs.stanford.edu

Abstract

Line Integral Convolution (LIC), introduced by Cabral and Leedom in Siggraph '93, is a powerful technique for imaging and animating vector fields. We extend the LIC paradigm in three ways:

1. The existing technique is limited to vector fields over a regular Cartesian grid. We extend it to vector fields over parametric surfaces, specifically those found in curvilinear grids, used in computational fluid dynamics simulations.

2. Periodic motion filters can be used to animate the flow visualization. When the flow lies on a parametric surface, however, the motion appears misleading. We explain why this problem arises and show how to adjust the LIC algorithm to handle it.

3. We introduce a technique to visualize vector magnitude as well as vector direction. Cabral and Leedom have suggested a method for variable-speed animation, which is based on varying the frequency of the filter function. We develop a different technique based on kernel phase shifts which we have found to show substantially better results.

Our implementation of these algorithms utilizes texture-mapping hardware to run in real time, which allows them to be included in interactive applications.

1. Introduction

Providing an effective visualization of a vector field is a challenging problem. Large vector fields, vector fields with wide dynamic ranges in magnitude, and vector fields representing turbulent flows can be difficult to visualize effectively using common techniques such as drawing arrows or other icons at each data point, or drawing streamlines[2]. Drawing arrows of length proportional to vector magnitude at every data point can produce cluttered and confusing images. In areas of turbulence, arrows and streamlines can be difficult to interpret.

Various techniques have been developed which attempt to address some of these problems. Max, Becker, and Crawfis[13], Ma and Smith[12], and Max, Crawfis, and Williams[14] have implemented systems which advect clouds, smoke, and flow volumes. These techniques show the flow on a coarse level but do not highlight finer details. Hin and Post[11] and van Wijk[16] have visualized flows with particle-based techniques, which show local aspects of the flow. Bryson and Levit [4] have used an immersive virtual environment for the exploration of flows. Helman and Hesselink[10] have generated representations of the vector field topology, which use glyphs to show critical points in the flow.

In this paper we discuss a new technique for visualizing vector fields which provides an attractive alternative to existing techniques. Our technique makes use of Line Integral Convolution (LIC)[5], which is a powerful technique for imaging and animating vector fields. The image of a vector field produced with LIC is a dense display of information, and flow features on the surface are clearly evident.

The LIC algorithm as presented by Cabral and Leedom in [5] is applicable only to vector fields over regular 2-dimensional Cartesian grids. However, the grids used in computational fluid dynamics simulations are often curvilinear. In this paper we show how to extend the LIC algorithm to visualize vector fields over parametric surfaces. Thus, for example, our extended algorithm allows us to visualize the flow over the surface of an aircraft or turbine.

In the original work on LIC, a technique for animation of vector field visualizations is presented. Our work extends this animation technique to apply to the parametric surfaces found in curvilinear grids as well.

Lastly, we present a new technique for displaying vector magnitude which can be applied to both 2-dimensional regular grids and parametric surfaces. Our method varies the speed of the flow animation to give an intuitive representation of vector magnitude.

In the next section we discuss the basic LIC algorithm. In section 3 we describe our extension to curvilinear surfaces. In section 4, we discuss the implementation of animation for curvilinear grid surfaces. In section 5, we introduce our technique for displaying vector magnitude. In section 6, we describe our implementation of all the algorithms in the paper. We conclude with a brief discussion of directions for further applications of the LIC algorithm in vector field visualization.

2. Background

The Line Integral Convolution (LIC) algorithm takes as input a vector field lying on a Cartesian grid and a texture bitmap of the same dimensions as the grid, and outputs an image wherein the texture has been “locally blurred” according to the vector field. There is a one-to-one correspondence between grid cells in the vector field, and pixels in the input and output image. Each pixel in the output image is determined by the one-dimensional convolution of a filter kernel and the texture pixels along the local streamline indicated by the vector field, according to the following formula:

$$C_{out}(i,j) = \sum_{p \subset \tau} C_{in}(p) \cdot h(p)$$

where
τ = the set of grid cells along the streamline within a set distance *0.5 l* from the point (i, j), shown as the shaded cells in Figure 1.
l = the length of the convolution kernel
$C_{in}(p)$ = input texture pixel at grid cell p

$$h(p) = \int_{\alpha}^{\beta} k(w)\, dw$$

where
α = the arclength of the streamline from the point (i, j) to where the streamline enters cell p
β = the arclength of the streamline from the point (i,j) to where the streamline exits cell p
k(w) = the convolution filter function

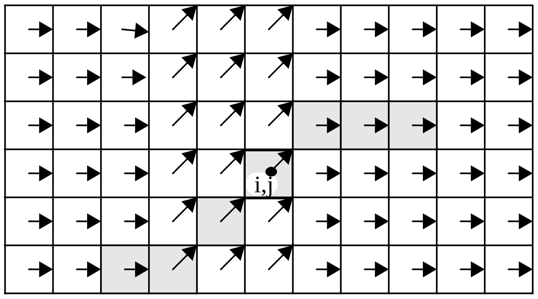


Figure 1.A vector field where the streamline through point (i,j) is shaded.

Thus each pixel of the output image is a weighted average of all the pixels corresponding to grid cells along the streamline which passes through that pixel’s cell. Section 4 of [5] provides the complete details of the algorithm. When this algorithm is applied at every pixel, the resulting image appears as if the texture were “smeared” in the direction of the vector field.

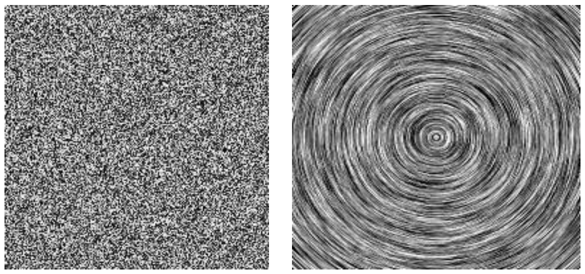


Figure 2.The input white noise bitmap on left is smeared using LIC on a circular vector field to produce the output image on the right.

3. Curvilinear Grid Surfaces

Because of the one-to-one correspondence between grid cells and pixels in the input/output images, the algorithm described above requires that the vector field lie on a regular, Cartesian grid. Here we show how to use the algorithm on 2-dimensional slices of structured curvilinear grids, which describe parametric surfaces.

We denote the curvilinear space coordinates of a point as $\vec{\xi} = (\xi, \eta, \zeta)$ and the physical space coordinates as $\vec{x} = (x, y, z)$. The vector which describes the velocity of the flow at each point is

$$\frac{\partial \vec{x}}{\partial t} = \begin{bmatrix} \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial t} \end{bmatrix}$$

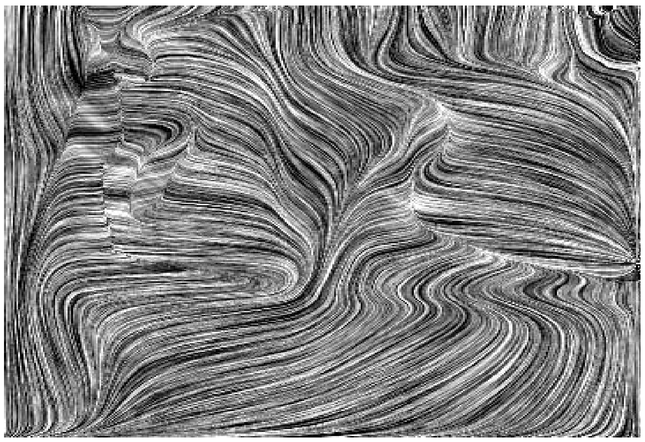
We transform the vector field to the coordinate system of the curvilinear grid, hereafter called “computational space.” The transformation from physical space to computational space is performed by multiplying the physical-space velocity vectors by the inverse Jacobian matrix s.t.

$$\begin{bmatrix} \frac{\partial \xi}{\partial t} \\ \frac{\partial \eta}{\partial t} \\ \frac{\partial \zeta}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial t} \end{bmatrix}$$

The computational-space vectors give velocity in grid-cells per unit time. Because the data points are given for integer coordinates in computational space, this constitutes a regular Cartesian grid.

We can compute a LIC-image of any 2-dimensional slice of the grid by projecting the vector field onto it. For example, if we want to examine the k=1 plane of the computational grid, which in many CFD data formats usually lies on the surface of the object about which the flow is being simulated, we drop the $(\partial\zeta/\partial t)$ term and use the 2-dimensional vector $[\partial\xi/\partial t, \partial\eta/\partial t]^T$ in the LIC algorithm.

The resulting image, which is a visualization of the vector field in computational space (see Figure 3) is then mapped onto the surface in physical space using a standard inverse mapping algorithm, such as that described in [9]. The inverse mapping converts the vector field representation back into physical space (Figure 3). The final result is a visualization of the flow which is dense, easily interpreted, and effectively handles the complicated areas of the flow.



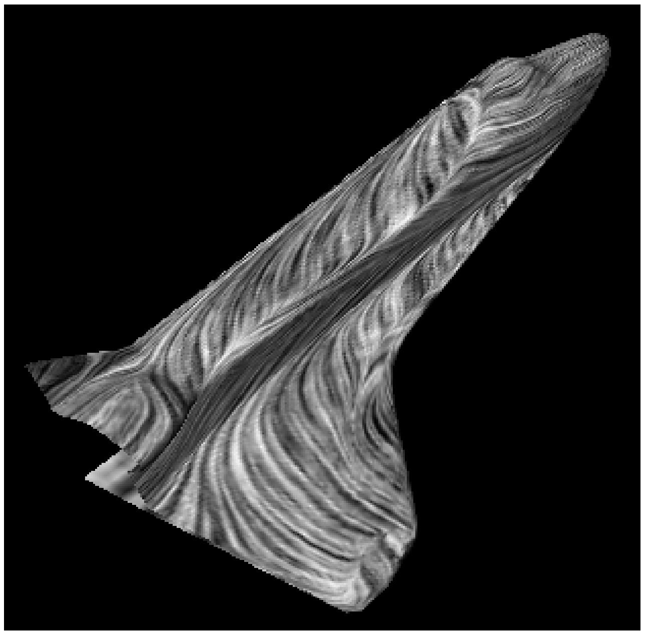


Figure 3. Top: a LIC image of the computational-space velocity field over the surface of the space shuttle. The input texture is white noise, and the convolution kernel is a simple box filter. Below: the LIC image above texture mapped over the space shuttle in physical space. Note the separation apparent on the fuselage and the vortices at the wingtip.

4. Animation

While the image described above and shown in Figure 3 correctly shows the streamline direction of the vector field, the visualization is ambiguous in regards to whether the flow is moving forward or backward along the lines indicated. To disambiguate the direction of flow, animation is useful. Also, animating a flow visualization is physically meaningful.

As Cabral and Leedom discuss in [5], periodic motion filters [6] can be used together with LIC to create the impression of motion, such that a flow appears to be moving in the direction of the vector field. A small number *n* of LIC images are computed, where in frame *i* the filter kernel is phase shifted by *is/n*, where *s* is the period of the filter function. When played back, these images cause the appearance of ripples moving in the direction of the vector field. Because the filter kernel is periodic, the *n* frames can be cycled through continually for smooth motion.

On a parametric surface, the images are ‘played’ by texture mapping each in turn onto the surface. However, additional steps must be taken to ensure that the animation does not introduce misleading information into the visualization. The conversion from computational space to physical space maps square grid cells into quadrilaterals of varying dimensions. Therefore, the length of the convolution filter, which is measured in computational space units, is mapped to varying lengths in physical space. The length of the periodic filter determines the size and speed of the “ripples” in the animation. The speed is given by the amount of phase shift in physical space per unit time. Thus, if the period of one filter function is longer in physical space than another, that ripple appears to move faster than a shorter filter.

As a result of the warping that occurs in the mapping from computational to physical space, the animation appears uneven and erratic. In areas where the grid is sparse, the flow appears as little ripples moving fast, because the convolution kernel has been compressed, and in areas where the grid is dense, the flow appears as large ripples moving slowly. Since there is no correlation between apparent speed and actual speed of the flow, this motion is highly misleading.

The situation can be corrected by varying the length of the convolution filter while computing the LIC image. The length of the convolution filter must vary inversely with the grid density in the direction of the flow. Where the grid is sparse in physical space, we want to use a narrow convolution filter in computational space, as it will be stretched out when mapped. Likewise, where the grid is dense in physical space, we want to use a wide filter in computational space, as it will be compressed when mapped. See figure 4, next page.

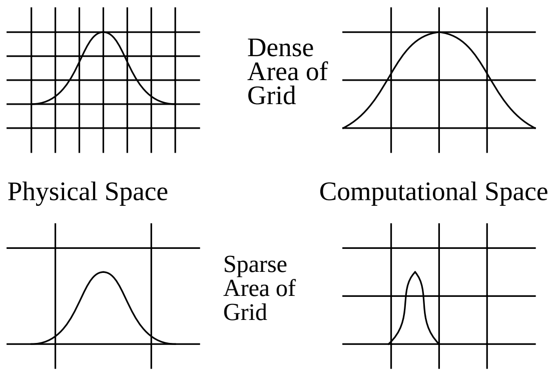


Figure 4. A convolution filter function in physical space and computational space. The goal is for the filter to be of uniform length in physical space, regardless of grid density. Therefore we stretch the filter in computational space in areas where the grid is dense, and compress the filter in computational space in areas where the grid is sparse.

We compute frames where the length of the convolution kernel used in the LIC algorithm at each grid cell *p* is given by

$$l(p) = a + \frac{b}{r(p)}$$

where

a is the minimum length of the kernel, measured in grid cells
b controls the range of possible kernel lengths, and
r is the grid density at grid cell *p* in the direction of the flow.

a must greater than 1; if the length of the filter is 1 or less, the LIC algorithm simply returns the input texture pixel, unaffected by the vector field.

b must be set to a finite length which will vary with the particular grid.

r(p) for each grid cell is given by

$$\left| \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}^{-1} \bullet \frac{\frac{d\vec{x}}{dt}}{\left|\frac{d\vec{x}}{dt}\right|} \right|$$

r(p) for the entire grid is computed by the following steps:

1) Normalize the vector field to unity in physical space.
2) Convert to computational space using the inverse Jacobian as described in section 3.
3) Take the magnitude of the computational-space vectors.

Figure 5 shows a single texture frame of an animation sequence computed in this way. When the 10 texture frames are played back, the flow appears smooth and even everywhere on the surface, rather than uneven and erratic. Thus we are able to use periodic motion filters even on parametric surfaces from curvilinear grids.

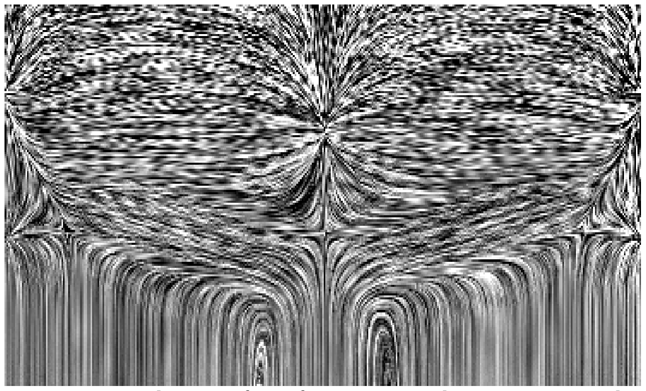
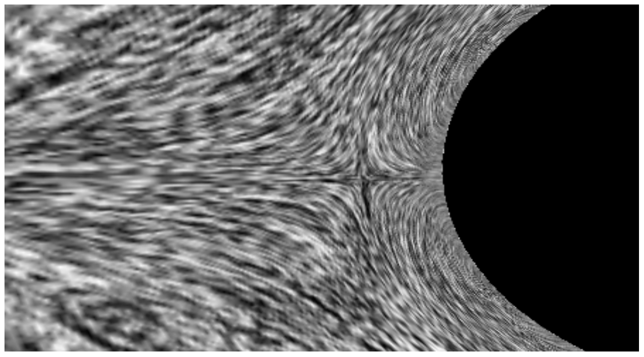


Figure 5a. A single texture frame from an animated sequence computed using the LIC algorithm with a raised cosine filter. The length of the filter varies inversely with the grid density in the direction of the flow. Therefore the ripples appear stretched in some areas and compressed in others. Details in section 4.



5b. Close-up of the texture frame from above mapped onto the curvilinear grid, used to simulate flow around a post. The bottom edge of the texture maps around the post. When mapped onto the grid, the ripples are of uniform size. When animated, the ripples give the impression of smooth flow

5. Variable Speed

The next step in flow animation, whether on a regular grid or on a parametric surface, is to give a visualization of vector magnitude as well as vector direction. Thus, in a CFD flow visualization, the periodic motion should be slow where the flow has low velocity and quick where the flow has high velocity. Cabral and Leedom [5] suggest achieving this effect by varying the frequency of the filter function, while keeping its length constant. However, the limited dynamic range (experimentation shows only between 2 and 4 ripples per kernel are interpretable) and the artifacts caused by changing the shape of the filter make it difficult to use this approach for meaningful results. We have found that a better solution is to vary the amount of filter function phase shift at each grid cell in proportion to the physical-space vector magnitude.

The amount of phase shift is what determines the apparent speed, given a uniform-length filter kernel. An infinitesimally small phase shift will appear not to move at all. Likewise, a 90-degree phase shift in every frame will produce a full cycle in four frames, which appears to move very quickly. (At anything greater than 180 degrees, temporal aliasing occurs.) Phase shifts ranging from 0 to 90 degrees can be mapped to the actual range of physical vector magnitude for a convincing variable-speed animation.

In a frame from a variable-speed animation sequence, each pixel will be computed with a convolution kernel that has a phase shift proportional to the corresponding grid cell's physical vector magnitude. Therefore the period of the filter function is different at each pixel, and there is no fixed number of frames that can be used in a cyclic animation. Therefore, we adopt the following strategy of sampling the "real" solution and interpolating to find the pixel values which we will display.

In practice, the texture frames are computed as follows:

1) First compute *N* LIC images, such that in image i, where θ_i is the amount of filter phase shift, $\theta_t = is/N$. As in section 4 *s* is the period of the filter function. The larger *N*, the more accurate the visualization. The intensity of pixel *p* in image *i* is defined as *T(i, p)*.

2) For each grid cell *p* , let

$$q = \frac{y - min(y)}{max(y) - min(y)}$$

where *y* denotes physical vector magnitude at grid cell *p*. *q* is a real number in [0,1] that gives the vector magnitude in cell *p* relative to the magnitudes in the whole grid.

3) The intensity of pixel *p* in frame *j* of the displayed image, *I(j,p)*, is found by interpolating linearly between the two LIC images from step (1) closest to it:

Let

$$\alpha = q\frac{N}{4}j\boldsymbol{mod}N - \left\lfloor q\frac{N}{4}j \right\rfloor \boldsymbol{mod}N$$

at cell p. Then

$$I(j,p) = (1-\alpha)\,T(\left\lfloor q\frac{N}{4}j \right\rfloor \boldsymbol{mod}N, p)$$

$$+ \alpha T(\left\lceil q\frac{N}{4}j \right\rceil \boldsymbol{mod}N, p)$$

6. Implementation

We use the texture-mapping capabilities of a high-end workstation [8] to display surfaces with LIC-images mapped onto them in an interactive program. All LIC images are computed prior to running the interactive program, since the LIC algorithm is fairly compute-intensive (images take on the order of several seconds to minutes to compute). The hardware is capable of switching between pre-loaded textures quickly enough that we are able to run animation in real time, while the user manipulates the surface.

For single-speed periodic motion, we find that 5 - 12 texture frames is sufficient for smooth animation.

For variable-speed animation, we are no longer able to precompute a finite number of frames and cycle through them, because the amount of phase shift varies at every grid cell. However, steps 2 and 3 of the algorithm described in section 5 are not compute intensive, once the *N* LIC-images of step 1 have been precomputed. This implies that a real time implementation of variable-speed animation should be possible. Unfortunately, we have not been able to achieve this with our hardware, an SGI Reality Engine, because the time required to load a new texture into the texture cache is too long to permit good frame rates. However, we expect that within the foreseeable future texture-mapping hardware will allow fast texture definition and this feature will be possible.

In the meantime, we have experimented with two alternative solutions.

(1) Calculate and store a large number of texture frames using a real phase shift at every grid cell which is a linear function of the physical vector magnitude in that grid cell. The number of frames required for anything more than a few seconds of animation using this solution is so large that storage requirements quickly exceed the memory capabilities of a workstation. Therefore either (a) the short sequence of animation must be continually restarted, which causes the flow to appear to "jump" every few seconds, or (b) the animation must be stored on video or another digital playback device, and the real time interactivity possible with all other techniques described in this paper are forfeited. We show an implementation of (a). The animation is smooth in spurts of 5 seconds, and the speed of the flow is clearly varying across the surface (see video).

(2) Approximate the continuous solution by choosing a minimum phase shift, ϕ, and quantizing all phase shifts as integer multiples of ϕ. In this solution, only $M = s/\phi$ frames need to be precomputed, because the *M* frames will form a complete cycle. The drawback of this solution is that it is susceptible to aliasing and rasterization caused by the sampling and quantization of phase shifts. The advantage is

that it can be played continually in real time on a workstation, without the jumps of solution (1).

To compute the frames for this discretized solution, we follow the same steps as described for the continuous solution in section 5, but round *q* to the nearest multiple of 1/*M*. In this case there is no interpolation and *M* frames suffice to form a cycle of animation.

In our implementation of solution (2), we see that while the speed of the flow is clearly varying, aliasing and rasterization artifacts do appear.

7. Future Work

There are several promising directions for future work. First, in order for this technique to become useful for practical applications, a number of extensions must be implemented. Foremost among these are multigrid solutions, unsteady flows, and unstructured grids.

Also, we hope to extend this technique to the visualization of 3-dimensional vector fields. While the LIC algorithm in itself extends easily to a 3-dimensional Cartesian grid, the output image data requires additional processing before a useful image is produced. Cutting planes, isosurfaces, or volume rendering techniques will be necessary for this extension. Experimentation with input textures and convolution filters will be needed to achieve effective images. Furthermore, new algorithms will be required to handle curvilinear grids in this situation as well.

8. Summary

We have presented several extensions to Line Integral Convolution. First, we have described how to use the LIC algorithm on curvilinear grid surfaces. We have shown how to solve the problems that arise when using periodic motion filters in LIC on a curvilinear surface. Lastly, we have introduced a method of incorporating visualization of vector magnitude into the LIC algorithm, by showing the animation at variable speeds. All algorithms are designed such that with modern graphics hardware the surfaces can be displayed, animated, and manipulated in real time.

Our visualization technique provides intuitive and accurate information about the vector field, and thus is a useful complement to other visualization techniques.

Acknowledgments

The author wishes to thank David Yip, who provided support of many forms throughout this research, the Applied Research Branch staff of NASA Ames, who offered helpful discussions, Marc Levoy, who gave advice on the research and on the writing of this paper, and Brian Cabral and Casey Leedom, who made their LIC code publicly available.

References

[1] D. Asimov, “Notes on the Topology of Vector Fields and Flows,” NASA Ames Research Center Report RNR-93-003, February, 1993

[2] M. Bailey, C. Hansen, *Introduction to Scientific Visualization Tools and Techniques*, Course Notes, ACM SIGGRAPH (1993).

[3] G. Bancroft, F. Merritt, T. Plessel, P. Kelaita, R. McCabe, A. Globus, “FAST: A Multi-Processing Environment for Visualization of CFD,” *Proc.Visualization ‘90*, IEEE Computer Society, San Francisco (1990).

[4] S. Bryson, C. Levit, “The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows,” *Proc. Visualization ‘91*, IEEE Computer Society, San Diego (1991).

[5] B. Cabral, C. Leedom, “Imaging Vector Fields Using Line Integral Convolution,” *Computer Graphics Proceedings ‘93*, ACM SIGGRAPH (1993)

[6] W.T. Freeman, E.H. Adelson, D.J. Heeger, “Motion Without Movement,” *Computer Graphics Proceedings* ‘91, ACM SIGGRAPH (1991)

[7] A. Globus, C. Levit, T. Lasinski, “A Tool for Visualizing the Topology of Three-Dimensional Vector Fields,” *Proc.Visualization ‘91*, IEEE Computer Society, San Diego (1991).

[8] *Graphics Library Programming Guide, Volume II*, Silicon Graphics, Inc. (1992)

[9] E. Haines, “Essential Ray Tracing Algorithms”, Chapter 2 in A. Glassner,Ed., *An Introduction to Ray Tracing*, Academic Press (1989)

[10] J. Helman, L. Hesselink, “Surface Representation of Two- and Three- Dimensional Fluid Flow Topology,” *Proc.Visualization ‘90*, IEEE Computer Society, San Francisco (1990).

[11] A. Hin, F. Post, “Visualization of Turbulent Flow with Particles,” *Proc. Visualization* ‘93, IEEE Computer Society, San Jose(1993).

[12] K-L. Ma, P. Smith, “Virtual Smoke: An Interactive 3D Flow Visualization Technique,” *Proc. Visualization* ‘92, IEEE Computer Society, Boston (1992).

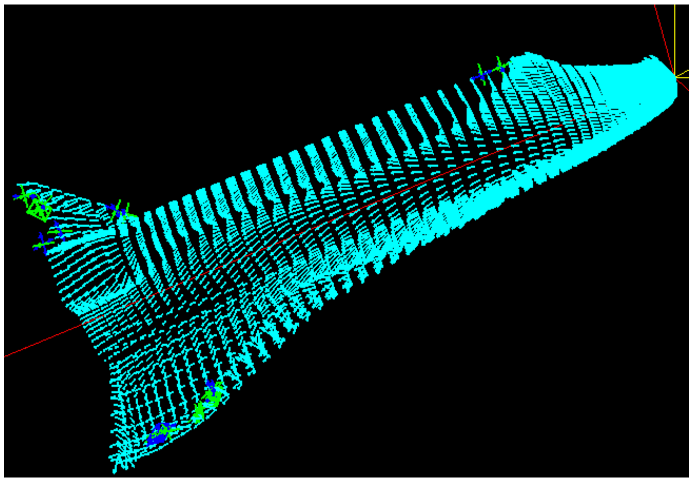
[13] N. Max, B. Becker, R. Crawfis, “Flow Volumes for Interactive Vector Field Visualization,” *Proc. Visualization* ‘93, IEEE Computer Society, San Jose(1993).

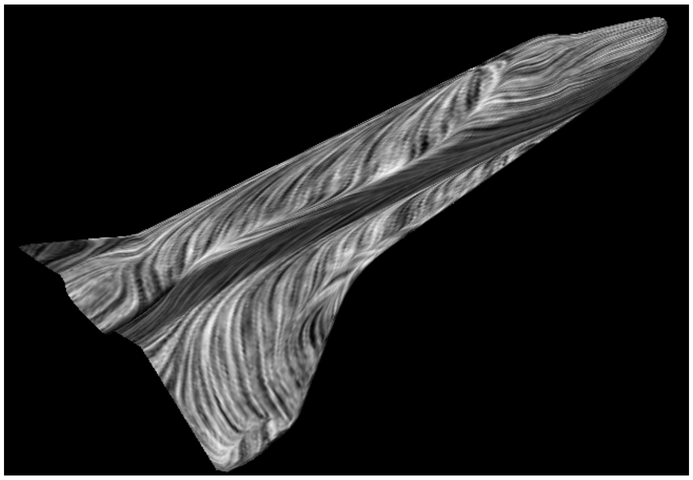
[14] N. Max, R. Crawfis, D. Williams, “Visualizing Wind Velocities by Advecting Cloud Textures,” *Proc. Visualization* ‘92, IEEE Computer Society, Boston (1992).

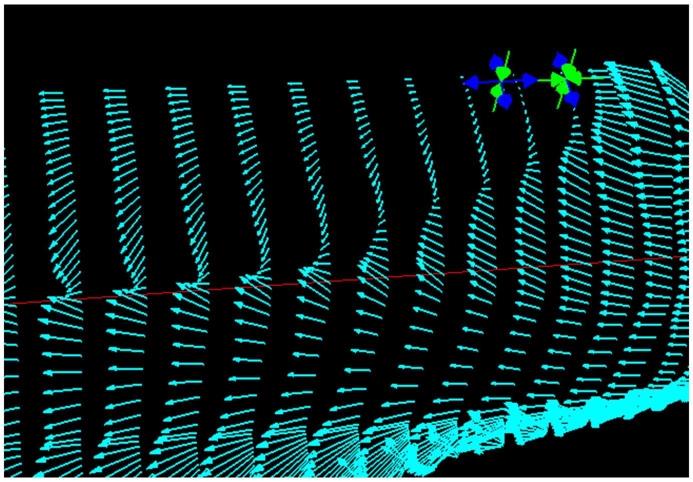
[15] P. P. Walatka, P. G. Buning, *PLOT3D User's Manual*, NASA Technical Memorandum 101067, NASA Ames Research Center.

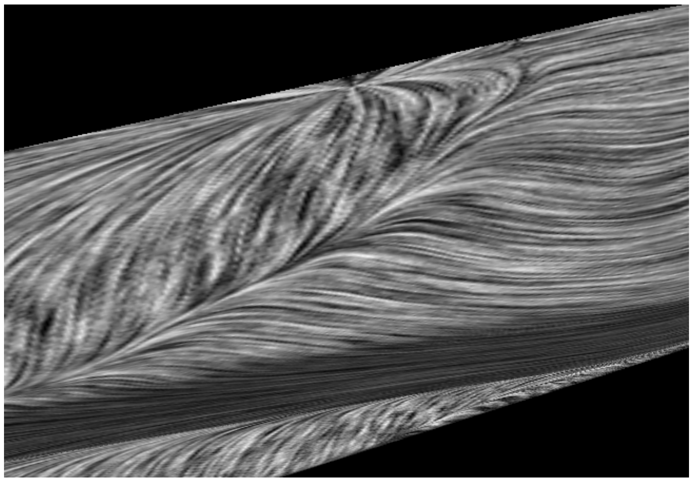
[16] J. van Wijk, "Rendering Surface-Particles," *Proc. Visualization '92*, IEEE Computer Society, Boston (1992).

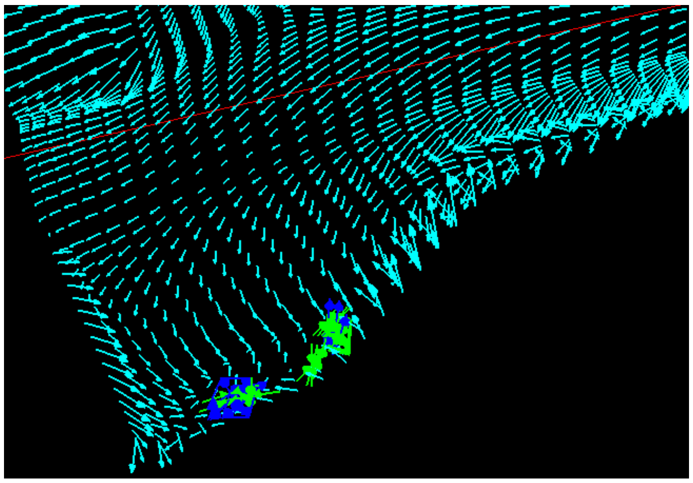
[17] W.H. Press, et al., *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge (1988).

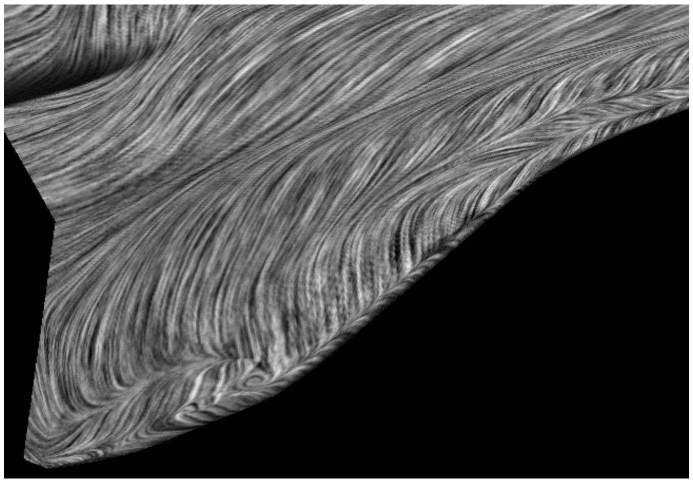

1a. The flow over the surface of the space shuttle visualized in FAST with arrow icons for velocity vectors and glyphs for critical points in the topology.


1b. The flow over the surface of the space shuttle visualized using Line Integral Convolution on the computational space vector field and texture mapped onto the shuttle surface.


2a. Detail of the top of the fuselage of the space shuttle, visualized in FAST.


2b. Detail of the top of the fuselage of the space shuttle, visualized using LIC for curvilinear surfaces.


3a. Detail of the wing of the shuttle.


3b. Detail of the wing of the shuttle.