Visualizing 3D Velocity Fields Near Contour Surfaces

Nelson Max
Roger Crawfis
Charles Grant
Lawrence Livermore National Laboratory
Livermore, California 94551

Abstract

Vector field rendering is difficult in 3D because the vector icons overlap and hide each other. We propose four different techniques for visualizing vector fields only near surfaces. The first uses motion blurred particles in a thickened region around the surface. The second uses a voxel grid to contain integral curves of the vector field. The third uses many antialiased lines through the surface, and the fourth uses hairs sprouting from the surface and then bending in the direction of the vector field. All the methods use the graphics pipeline, allowing real time rotation and interaction, and the first two methods can animate the texture to move in the flow determined by the velocity field.

Introduction

There are many representations of velocity fields: streamlines, stream surfaces, particle traces, simulated smoke, ..., etc. One of the simplest is to scatter vector icons, for example, small line segments, throughout the volume. There are two fundamental problems with such an approach. First, the 2D projection of a line segment is ambiguous; many 3D segments can have the same projection. Second, densely scattered icons can overlap and obscure each other, leading to a confusing image. These problems can be partially solved with real time (or playback) animation, since motion parallax can resolve the projection ambiguities. In addition, icon motion in the velocity direction can give added visual information.

The second problem can also be resolved by restricting the icons to special regions of interest. For example, in [Crawfis92] and [Crawfis93], the vectors' opacity depended on their magnitude, so only the regions of highest velocity were emphasized. In this paper we take the region of interest to be on or near a contour surface. The scalar function being contoured can come directly from the vector field, for example, the vector magnitude or a vector component. It can also be an independent scalar field defined on the same volume, for example, porosity in a flow simulation, or a linear or quadratic function for interactive slicing. We report here on four different techniques for visualizing velocity fields near contour surfaces.

Spot Noise

Van Wijk [vanWijk91] generated a directional texture by superimposing many oriented shapes such as ellipses. For texture on curved surfaces, the texture plane is mapped to the surface, and the texture generation accounts for the stretching induced by the mapping. Van Wijk visualized a tangential velocity field on a ship hull with this method. Cabral [Cabral93] has generated similar but more accurate 2D velocity textures by "Line Integral Convolution" of random noise, and Forssell [Forssell94] extended this with mapping to visualize velocity near an airplane surface. Both of these techniques can be animated to make the texture flow. However, they are not applicable to contour surfaces, which cannot easily be parameterized.

Stolk and van Wijk [Stolk92, vanWijk93] have also visualized flows with surface particles: individual spots motion-blurred to elliptical shapes and composited separately onto the image in software. Each spot has a surface normal, which is carried along appropriately by the flow, and used in the shading. These particles can also move in animation, but only on surfaces related to the flow, not on fixed contour surfaces of an unrelated function. Here we use the same sort of spots, but take advantage of hardware rendering for interactive speed. We do not restrict the particles to lie on a surface, and therefore do not use normals in the shading. Instead, we think of the particles as spheres, which are motion blurred to ellipsoids. We only draw particles which lie within a specified distance D of the contour surface of interest.

We should emphasize that most of the previous work represents tangential flows on a surface, for example, a stream surface, but the problem we are trying to address

uses contour surfaces of functions possibly unrelated to the velocity vectors, which are therefore not usually tangent to the surface.

We assume that the vector field $V(x, y, z)$ and the scalar function $f(x, y, z)$ are defined on the same rectilinear lattice. To estimate the distance $d(x, y, z)$ of a particle at (x, y, z) from the contour surface $f(x, y, z) = C$, we use a technique described by Levoy [Levoy88]. We approximate the gradient $\nabla f(x, y, z)$ by finite differences of the neighboring vertex values of f, and store the magnitude $|\nabla f(x, y, z)|$. Then

$$d(x, y, z) = \left| \frac{f(x, y, z) - C}{|\nabla f(x, y, z)|} \right|$$

where the quantities $f(x, y, z)$ and $|\nabla f(x, y, z)|$ are trilinearly interpolated from the eight surrounding lattice vertices.

We wish to randomly deposit particles uniformly into the region R where $d(x, y, z) < D$. Assuming the specified distance D is larger than the cell size in the lattice, the following procedure does this efficiently. We mark all lattice vertices which are within D of the contour surface, and then mark all cells which have at least one marked vertex. These markings are updated whenever the user changes C or D. We randomly insert a specified number N of particles in each marked cell, and render a particle at (x, y, z) only if $d(x, y, z) < D$.

The particles are kept in a linked list, which also contains their ages. They are moved in each time step by second order Runge-Kutta integration, using velocities trilinearly interpolated from the lattice vertices. If a particle moves into an unmarked cell or out of the data volume, it is scheduled for deletion from the linked list. At each time step, we also check that all marked cells still have N particles, and add or delete particles as necessary.

The particle size is proportional to a factor $b = s(d)$, which is equal to 1 for small d and decreases continuously to zero when d reaches D. Thus the particles fade in as they first cross into R, and fade out as they leave. Since a particle could randomly be created near the contour surface to replace another particle leaving a cell, we actually take $b = \text{minimum}(s(d), ka)$, where k is a constant and a is the age of the particle. This makes new particles fade in at birth. They are similarly faded out when deleted.

As in [vanWijk93], we draw the particles as blurred ellipses, stretched out in the direction of motion. However we do the compositing using the hardware in our SGI workstations. A single blurred disk is used as the texture, and is mapped to a 3D rectangle. If r is the radius of the particle, P is its position vector relative to the viewpoint, and V is its velocity vector, then the rectangle vertices are at $P + S + T$, $P + S - T$, $P - S - T$, and $P - S + T$, where

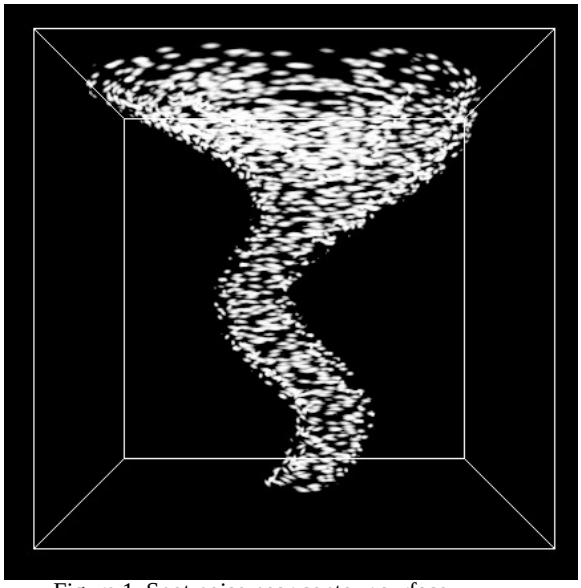


Figure 1. Spot noise near contour surface

$$S = r\frac{V \times P}{|V \times P|} \text{ and } T = V + r\frac{S \times P}{|S \times P|}.$$

Basically, T is along the velocity direction, but the second term is added so that the particle will shrink to a small round dot of radius r when the velocity approaches zero or is oriented near the viewing direction. Because these semi-transparent particles are sent through the graphics pipeline after the opaque objects, they can be combined with opaque contour surfaces in the z-buffer and be appropriately hidden. Currently they are all the same color, so they do not need to be sorted. (See [Max93].) Figure 1 shows a collection of ellipsoidal motion-blurred particles near a contour of velocity magnitude on a “tornado” velocity data set. Figure 2 includes a contour surface of velocity magnitude, which hides some of the dots. Figure 3 represents a 0.5 micron simulation of the airflow through a HEPA filter.

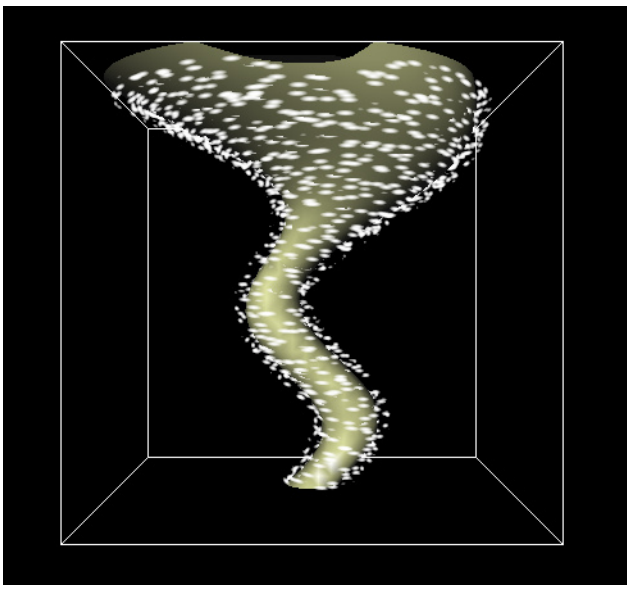


Figure 2. Spots noise with contour surface.

An animation has been produced showing the flow at varying velocity contours, from high to low. A contour region of moderate flows is illustrated here. Figure 4 illustrates a velocity contour near zero and close to the metallic fibers.

The implementation is in C++, using SGI's Inventor for the user viewing interface. The contour value *C* and width *D* are controlled by sliders. On our office workstations with Elan™ graphics, the actual texture mapping is done in software, but the same code automatically calls the hardware texture mapping on the Onyx™ workstation in our Graphics Lab. We can thus get real time rotation and particle motion on small data sets, and interactive performance on large data sets.

Particle traces on 2D surfaces

The goal of this technique is to represent the magnitude and direction of a vector field at each point on an arbitrary set of surfaces (not necessarily a contour surface), allowing long flowlines to be easily understood, but without having the visualization become too dense or too sparse at any point.

To do this we try to draw long, evenly spaced particle traces. The beginning and ends of these traces do not necessarily indicate a source or sink in the vector field. Traces begin and end over the entire surface in order to keep a nearly constant density of lines in the image. The particle trace lines are broken into small segments of contrasting color. The length of the segments represents the magnitude of the vector field (i.e. a constant time interval). The direction of the segments is the direction of the vector field at that point projected onto the 2D surface. In still pictures, using a sawtooth shaped color map across each segment resolves the directional ambiguity of the lines. We experimented with several different projections of the vector field to the surfaces.

Once the particle traces are calculated and rendered, color table animation can be used to add motion to the display so that the lines "flow" in the direction of the vector field with a velocity proportional to the magnitude of the vector field at each point. This flow animation was a primary goal of this technique. Color table animation is an old technique [Shoup79] which has been applied to flow visualization by Van Gelder and Wilhelms [VanGelder92].

The first step in this technique is to scan convert the surfaces into an octree. A piecewise linear representation of the 2D surface(s) is stored in the octree. Each cell holds a plane equation (four numbers). The octree allows us to use any kind of surface (as long as the surface does not pass through any leaf cell twice, which is unavoidable for self-intersecting surfaces). The octree allows fast access to adjacent cells but is much more memory efficient than a full 3D grid. Only those cells that intersect the surfaces are present in the octree. In this implementation, the surfaces are subdivided down to a constant size cell.

A seed point is then chosen to try to place the first particle trace. The particle is advected along the surface in both directions, forward and backwards, using a variable step size Euler's method. We continue to advect the particle until it reaches a stagnation zone, reaches an edge of the surface, or becomes too close to its own trace or that of another particle. The particle trace is considered acceptable if it is longer than the current length threshold. If the trace is too short, it is erased and a new seed point is tried. The length of the erased trace is preserved in the cells in which it passed. This value is used to prevent extensive recalculation while backtracking.

The seed points are chosen on an integer lattice in a spatially hierarchial manner so that the first particle traces will start well away from each other and are likely to be traced for long distances before getting too close to other lines. Random placement of seed points would also be likely to yield long lines for the first points chosen, but would not guarantee that some seed point was near every point on the surfaces. All seed points, at some particular resolution, are tried first using a large length threshold. Then the length threshold is reduced and the process is repeated. Gradually reducing the length threshold from some maximum to minimum produces the most esthetically pleasing distribution of particle traces, but takes longer to calculate than using a single length threshold value.

Projections

Four techniques were tried for projecting a 3D vector onto the 2D surface. Three techniques, normal, xy normal and cylinder projections, are viewpoint independent. The eye projection technique is viewpoint dependent. Viewpoint dependent techniques require the particle traces to be recalculated each time the viewpoint is changed, while viewpoint independent techniques do not require this recalculation. To compare these four projections and their ability to indicate the 3D flow, we have used a simple test case: a constant velocity field.

Normal Projection

With the normal projection technique, the 3D vector at a point on the surface is projected onto the surface in a direction parallel to the surface normal at that point. This projection, while being very straightforward, can yield very nonintuitive particle traces. Figure 5 shows the tornado surface in a vector field in which all vectors are in exactly the same direction, pointing to the upper right at 45

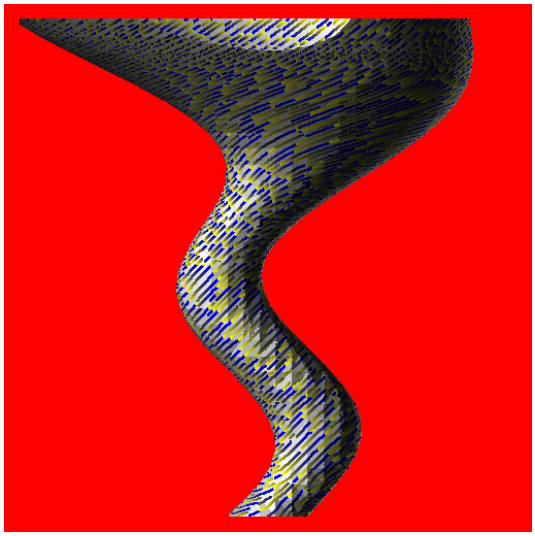


Figure 5. Normal Projection

degrees. The uniformity of the vector field is not at all apparent with this projection.

XY Normal Projection

The xy normal projection technique is designed for producing film loops where the viewpoint is rotated about the z (vertical) axis. In this technique the 3D vector is projected onto the surface in the direction of a vector which consists of only the x and y components of the surface normal at that point. As can be seen in figure 6, this preserves the z component of the vector field and gives a somewhat more intuitive visualization, but the uniformity of the vector field is still not readily apparent.

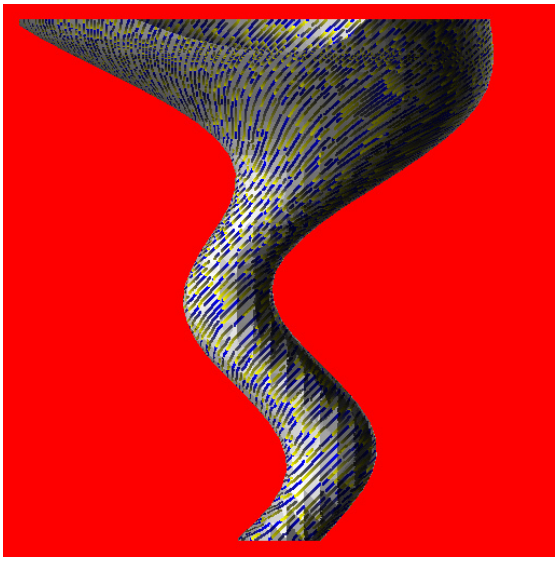


Figure 6. XY Projection

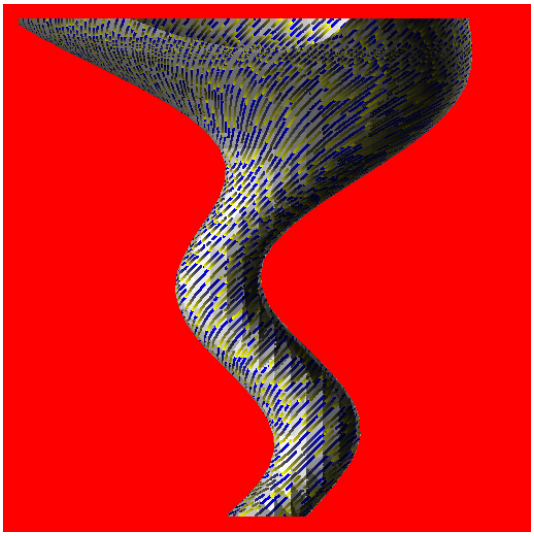


Figure 7. Cylinder Projection

Cylinder Projection

The cylinder projection technique is also designed for producing film loops where the viewpoint is rotated about the z (vertical) axis. In this technique the 3D vector is projected onto the surface in the direction of a vector which points away from the rotational axis. As can be seen in figure 7, this gives results very similar to the xy normal projection. This projection is only suitable for simple surfaces which are centered on the rotational axis.

Eye Projection

In the eye projection technique, the 3D vectors are projected onto the surfaces in a direction parallel to the viewing direction. As shown in figure 8, for a single image this technique produces the most intuitive representation of the constant direction vector field. The uniform direc-

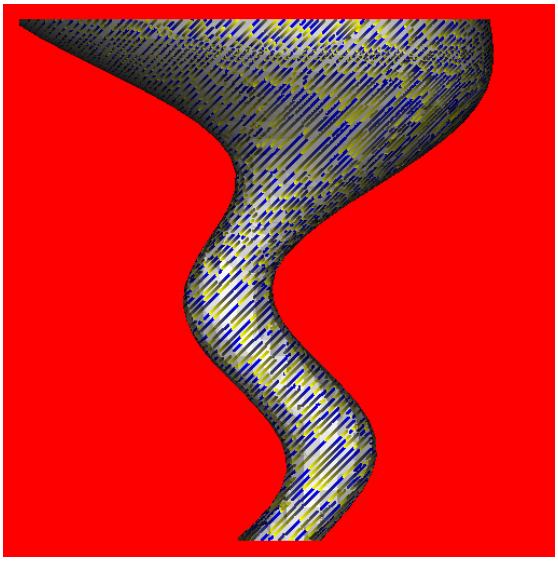


Figure 8. Eye Projection

tion of the vector field is readily apparent. When changing the viewpoint for a film loop, the projection of the 3D vectors to the surfaces changes, resulting in a different set of particle traces. We attempt to minimize the visual effect of these changes by using the same set of trial seed points as the previous frame, and by starting each particle trace with the same "phase" as a nearby trace in the previous frame. Figure 9. is a color reprint of Figure 6..

In conclusion, the uniformly spaced, animated particle traces are an effective means of visualizing a 2D vector field, but the projection of a 3D vector field onto a 2D curved surface loses and distorts some of the information from the 3D field, limiting the usefulness of these techniques for 3D. This is a difficulty for our goal of representing flows non-tangential to the surface. It is still an effective technique for stream surfaces or other tangential flows.

Line Bundles

Line bundles use the back-to-front compositing and overlapping of splatting to construct a volume of tiny line segments. Taken as a whole, these line segments construct the appearance of a fibrous volume. While drawing many tiny vectors to represent a vector field is not new, we have combined this idea with back-to-front compositing and techniques to generate anisotropic textures (see Figure 10). Our basic implementation plan is to extend the concept of splats - each data point is composited into the frame buffer in a back-to-front manner [Crawfis93]. Rather than trying to reconstruct a C^1 3D signal, we want a very discontinuous, yet antialiased, 3D volume representation. At each data point, a collection of antialiased lines is splatted. The lines are randomly scattered within a 3D bounding box associated with each splat. The hue, saturation, value and center position within the box are all randomly perturbed. The direction of each line is in the direction of the flow field. The jittering of the color and position produce a nice anisotropic texture oriented in the direction if the flow field, even when the lines are so dense that there is no space between them. The primary color about which we jitter can be different for each splat or can be fixed for the entire volume. Having different colors per splat allows us to encode additional information about the volume, either the vector magnitude, some separate scalar field, or a positional indicator. Using a single primary color allows us to precompute the line bundle into a GL object, which is then very rapidly reoriented and redrawn at each data point. Line bundles in excess of over 300 line segments can be used for each data point with no degradation in real-time performance.

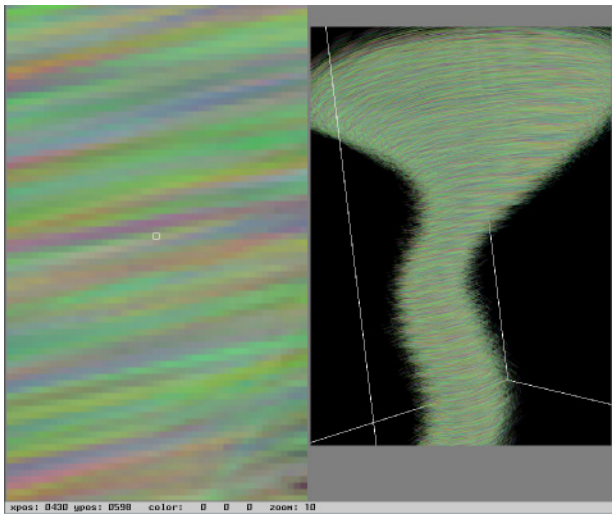


Figure 10. Line bundle tornado with magnification

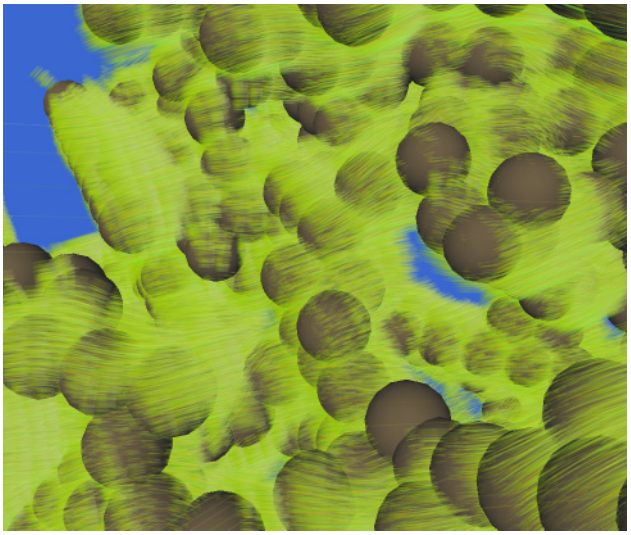


Figure 11 Line bundle near aerogel surface

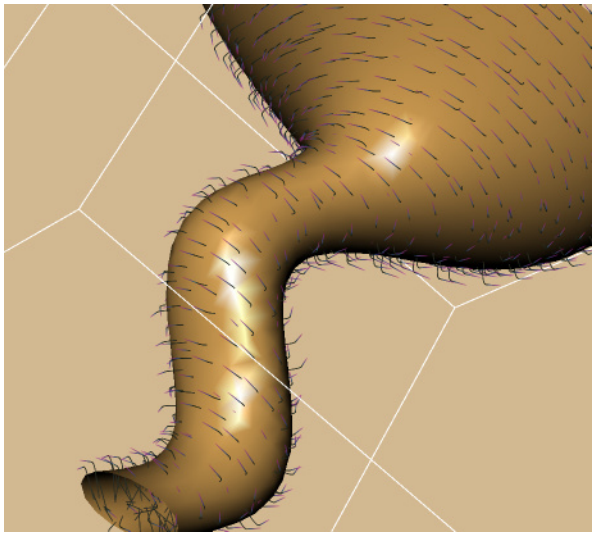
Three key issues are addressed in these fibrous volumes: back-to-front compositing for a thin wispy-like appearance, antialiased lines with transparent heads and tails to avoid unwanted edges, and controlled jittering of colors and positions to avoid regular patterns or completely filled regions. Figure 10 shows a sample tornado data set using a homogenous color that is heavily jittered. A zoomed in portion of the image is to the left of it. Notice the antialiasing and lack of any hard edges. Figure 12 shows the wind field over North America in a simulated global climate model. Data points close to the isocontour surface of a particular wind velocity magnitude were chosen for the line bundle splatting. Figure 11 represents the airflow through a filter substrate known as aerogel.The data points were chosen to lie close to the surface of the filter particles. This provides a nice mossy appearance. In Figure 13, points were chosen near a velocity contour of the HEPA filter simulation, and are color coded by velocity, giving a solid volume with a fibrous texture. These images can be generated in real time on a mid- to high-end graphical workstation.

Hairs

We tried to use the line bundles to represent the flow around or near a surface. This broke down when the flow was slightly into the surface. A solution is to grow tiny hairs coming out of the surface. We draw the line segments out of the surface first and then have them bend in the flow field, much like normal hair. Several controls over this behavior are offered. The physical layout of the hairs are specified by a number of connected line segments, by an interpolation of the normal vector and the velocity vector, and by a stiffness or weighting factor per segment. The default number of segments is six, and all of the images here were generated using only six segments. The interpolation is completely specified by the user with coefficients t_i at each line segment point. A new directional vector is generated using the formula: *Direction* = w_i * (t_i * *Normal* + (*1*-t_i) * *Velocity* * *Velocity_Scale*). The *Normal* vector is normalized, and the velocity is scaled by the user parameter *Velocity_Scale*. Smooth curves or sharp bends can be specified with the proper t_i's. The *Direction* vector is then added to the endpoint of the last line segment. The weighting by w_i is useful to control the apparent stiffness of the hair. Greater weights can be given to the initial segment in the direction of the normal, lower weights to the middle segments and finally, large weights can be given to the last one or two line segments to produce longer hairs in the velocity direction. This defines an individual hair. A number of hairs are scattered throughout the splat volume and jittered from one splat to the next. The splats are positioned near the contour surface by the method of Levoy discussed above. They are moved towards the contour surface by a multiple of the gradient.

The color and transparency of the hairs are controlled by a weighted function of the hair's root color, a specified splat color, a vector head color, and an HSV space jittering. The user specifies a root color and a vector head color. A splat color is derived from a scalar field to color table mapping. At each segment, the user can specify the fraction of the root color, the fraction of the splat color, and the fraction of the vector head color that the endpoint of that segment should be. A random jitter is added to this final color. The jittering is controlled by a scale factor for each component. All computations are performed in HSV space, with the hue wrapping around from one to zero, and the saturation and value components clamped at zero and one. A transparency value is also specified at each segment. No jittering is applied to this.

Figure 14 shows our tornado with very sparse and opaque hairs. With these settings you can clearly see the hairs coming from the normal direction and bending into the velocity direction. Figure 15 has more hairs that are much more transparent.



Conclusions

How do these techniques relate to each other and previously developed techniques? Table 1. correlates 3D vector field visualization techniques to various attributes and problem tasks. Our motto throughout this research is that we are not developing better techniques, but expanding on the set of tools available. Different scientists can gain insight better with different tools and many tools are usually required in an analysis. We have not tried to be all encompassing or thorough. There are many other techniques that should be added: stream tubes, topology extraction, shaded particles, etc. There are also many other characteristics that should be considered. Hopefully it is a useful starting point.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48, with specific support from an internal LDRD grant. Barry Becker helped with some of the programming and debugging, and Jan Nunes helped with the video production. Th HEPA filter data is courtesry of Bob Corey at LLNL. The Aerogel data is courtesy of Tony Ladd and Elaine Chandler, both at LLNL. We would also like to thank the reviewers for their valuable comments.

	Hedge-hogs	Particle Trace	Stream Line	Stream Ribbon	Flow Volume	Textured Splats	Stream Surface	LIC	Spot Noise	Line Bundles	Constrained Stream Lines	Hairs
Dimensionality	0D/1D	0D	1D	2D	3D	3D	2D/3D	2D/3D	3D	3D	2D	2D
Hardware Accelerated	●	●	●	●	●	●	❍	❍	●	●	●	◗
Doesn't Require Advection	●	❍	❍	❍	❍	●	❍	❍	❍	●	❍	●
Dynamic Motion	❍	●	❍	❍	◗	●	❍	●	●	❍	●	❍
Global Representation	◗	❍	❍	❍	●	●	◗	●	●	●	❍	❍
Near Surfaces	❍	❍	❍	❍	❍	◗	❍	◗	●	◗	●	●
Near Tangential Surfaces	❍	❍	❍	❍	❍	❍	●	●	●	●	●	❍
Unsteady Flows	❍	●	❍	❍	◗	●	❍	◗	❍	●	❍	◗
User Probing	❍	●	●	●	●	❍	◗	❍	❍	❍	❍	❍
Interactive Rendering	◗	●	●	●	●	◗	◗	❍	◗	●	❍	●

Table 1: Comparison of 3D Vector Field Visualization Techniques

References

[Cabral93] Brian Cabral and Lieth Leedom, "Imaging vector fields using line integral convolution," Computer Graphics Proceedings, Annual Conference Series, ACM Siggraph, New York (1993) pp. 263 - 270

[Crawfis92] Roger Crawfis and Nelson Max, "Direct volume visualization of three dimensional vector fields," Proceedings, 1992 Workshop on Volume Visualization, ACM Siggraph, New York (1992) pp. 261 - 266

[Crawfis93] Roger Crawfis and Nelson Max, "Texture splats for 3D scalar and vector field visualization," Proceedings, Visualization '93, IEEE Computer Society Press, Los Alamitos, CA (1993) pp. 261 - 266

[Forsell94] Lisa Forssell, "Visualizing flow over curvilinear grid surfaces using line integral convolution," these proceedings.

Levoy88] Mark Levoy, "Display of surfaces from volume data," IEEE Computer Graphics and Applications Vol. 8, No. 5 (May 1988) pp. 29 - 37

[Shoup79] Richard Shoup "Color table animation," Computer Graphics Vol. 13 No. 4 (August 1979) pp. 8 - 13

[Stolk92] J. Stolk and J. J. van Wijk "Surface particles for 3D flow visualization," in "Advances in Scientific Visualization," F. H. Post and A. J. Hin, eds., Springer, Berlin (1992) pp. 119 - 130

[VanGelder92] Allen Van Gelder and Jane Wilhelms, "Interactive animated visualization of flow fields," Proceedings, 1992 Workshop on Volume Visualization, ACM, New York (1992) pp. 47 - 54

[vanWijk91] J. J. van Wijk "Spot noise: texture synthesis for data visualization," Computer Graphics Vol. 25 No. 4 (July 1991) pp. 309 - 318

[vanWijk93] J. J. van Wijk "Flow visualization with surface particles," IEEE Computer Graphics and Applications Vol. 13 No. 4 (July 1993) pp. 18 - 24

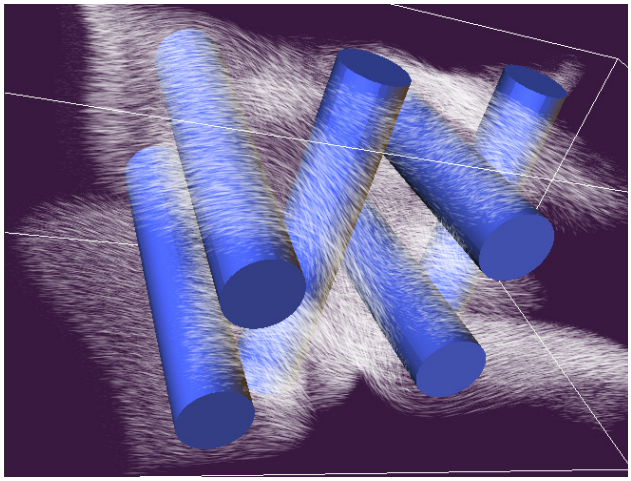


Figure 3.Spot noise rendering of HEPA filter..

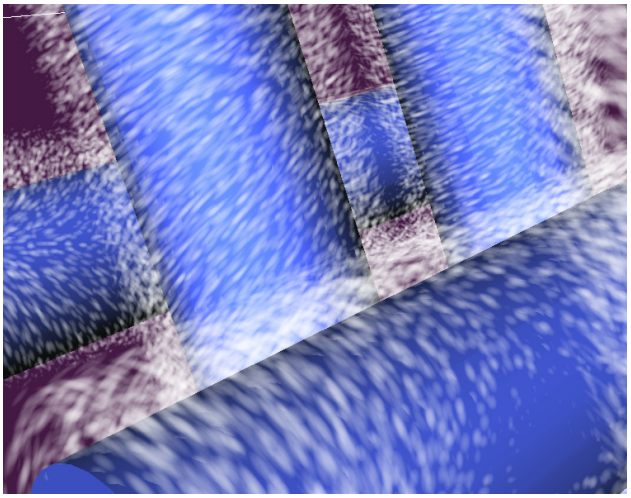


Figure 4. Spot noise near filter fibers.

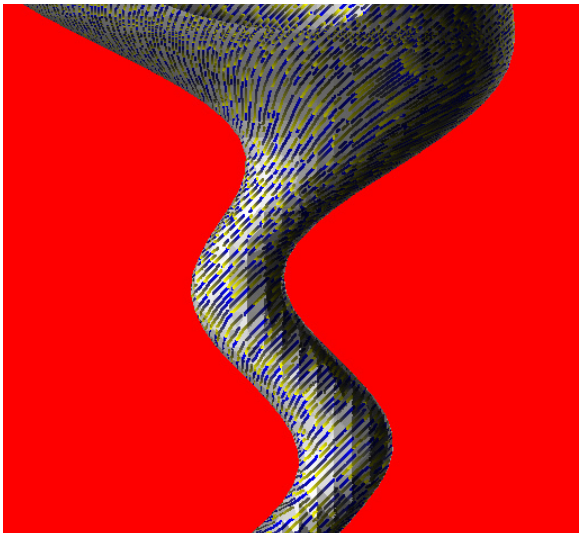


Figure 9. Projected Streamlines - XY Projection.

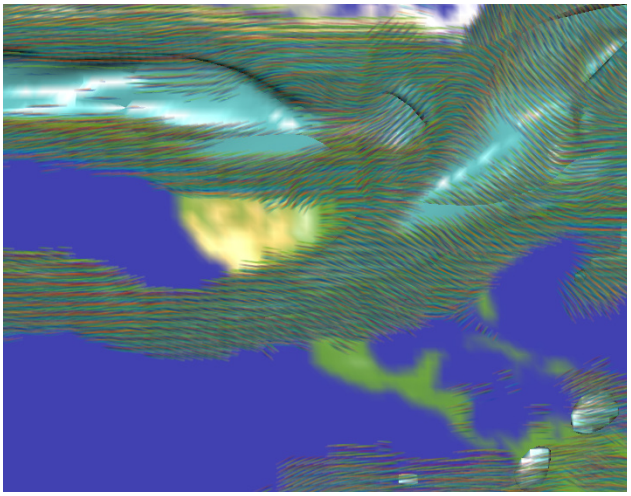


Figure 12. Line bundle near aerogel surface.

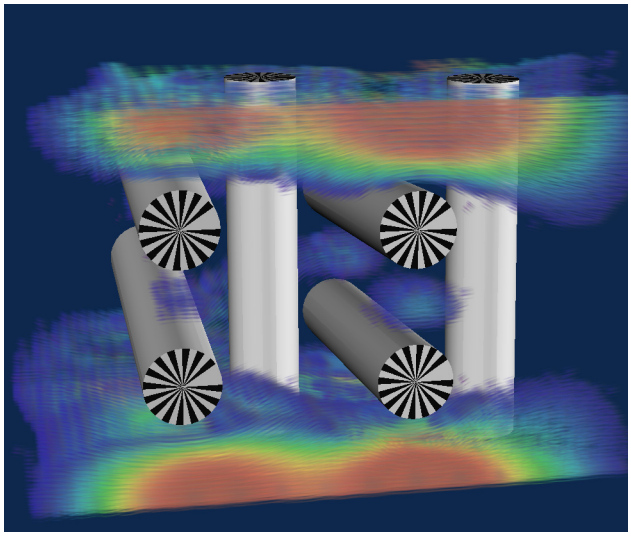


Figure 13. Line bundle rendering of HEPA filter.

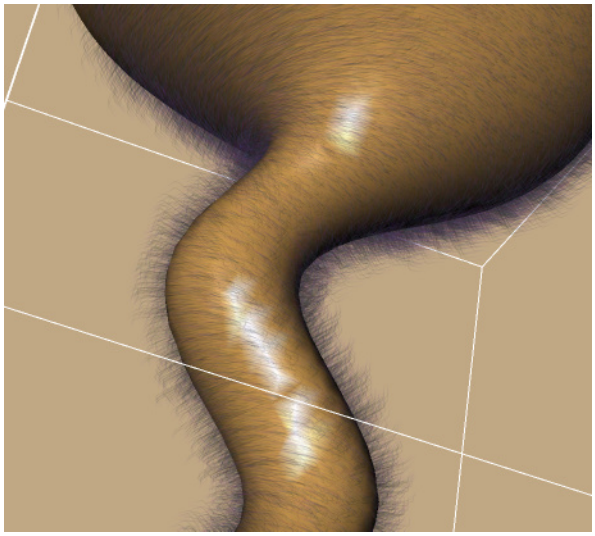


Figure 15. Finer hair on tornado velocity contour.