224 004

UCRL-JC-116498
PREPRINT

Vector Field Visualization

Roger Crawfis
Nelson Max
Barry Becker

This paper was prepared for submittal to
IEEE Computer Graphics and Applications Journal
August 1994

March 1994

Lawrence
Livermore
National
Laboratory

Vector Field Visualization

Roger Crawfis (crawfis@llnl.gov)
Nelson Max (max2@llnl.gov)
Barry Becker (beckerb@llnl.gov)

Lawrence Livermore National Laboratory
P.O. Box 808 / L-301
Livermore, CA 94551

Abstract

*Understanding 3D vector fields is a current challenge for scientific visualization. This paper describes two techniques useful for understanding vector fields. The first technique is an extension of stream lines, stream surfaces, and flow ribbons to a new construct which we call **flow volumes*** [1]. *With this technique, the user can interactively probe the vector field, releasing artificial smoke or dye and viewing its propagation by the flow field. This flow volume is constructed and volume rendered in a very efficient manner. The second technique attempts to represent all of the interesting areas of the three-dimensional flow, by extending the concepts of volume rendering to handle textured or anisotropic density clouds. These anisotropic density clouds are rendered efficiently using a new technique we call* **texture splats** [2].

Introduction

When studying complex phenomena such as the weather, fluid dynamics, electromagnetics, and ground water contamination, a primary attribute that needs investigating is a vector field: wind velocity, fluid vorticity, electrical and magnetic wave propagation, etc. Two prevalent paradigms exist for studying these flows: a hedgehog or arrow plot of some region of the flow fields; and the advection of a set of particles. Extensions to particle traces and streamlines have been developed over the past few years: stream polygons [3], spot noise [4], stream surfaces [5], [6] and iconic representations [7], [8]. A third paradigm has been established recently to examine the topology of flow fields. Hellman and Hesselink [9] have developed a technique to build stream surfaces to represent the topology of the flow. Globus et al. [10] have developed techniques to identify the critical points of a flow field. Iconic representations can be placed at these points with appropriate streamlines connecting them. Very little work has progressed on the basic hedgehog technique. Advanced techniques in this area have focused on generating a coherent texture to represent the overall flow: [11], [12].

Flow Volumes

Flow volumes extend the particle advection paradigm to generate a continuous three-dimensional volume rendered as a varying density cloud. It combines the adaptive refinement of Hultquist's stream surfaces [5] with the tetrahedral volume rendering of Shirley and Tuchman [13], by extending the stream surface to a volume and generating a consistent subdivision of the volume into tetrahedra.

Advection and Adaptive Subdivision

We start with a base polygon (currently a square), and generate streamlines from each corner. At each time step, a set of triangular prisms is generated from the previous time step's data points and the current time step's data points. If there is a large divergence in the streamlines comprising a triangular prism, more streamlines are added and a finer subdivision of the volume is generated. Some volume cells may have non-planar faces. These faces may become self-intersecting polygons when projected onto the picture plane and hence cause problems in the volume compositing scheme. While such problem cells may be rare in projecting a fixed curvilinear grid, they will be more common in flow volumes, since small scale variation in the velocity field can easily distort the faces. Therefore, we have chosen to decompose the volume into tetrahedra. A method for doing this consistently that avoids gaps or self-intersections across the non planar faces is presented in [1] .

Volume Rendering

We have developed a new volume rendering approximation which takes advantage of texture mapping and compositing hardware available on modern workstations. This allows tetrahedra to be composited by the Projected Tetrahedra algorithm of Shirley and Tuchman [13], without artifacts due to linear approximation of the non-linear opacity effects. The Shirley-Tuchman algorithm divides the projection of each tetrahedron into from one to four triangles, each bounded by the projections of the tetrahedron's edges. Figure 1 shows the two non-degenerate cases (where no vertex projects onto another vertex or edge), which require three and four triangles respectively.

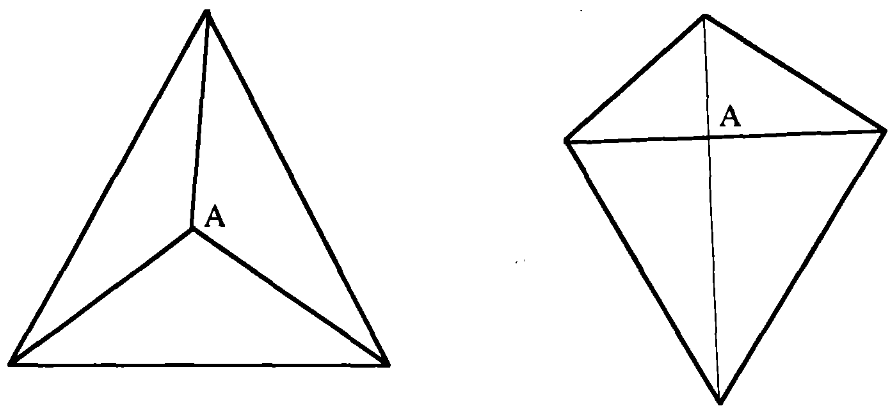


Figure 1. Common Tetrahedral Projections

The vertex marked A in each projection corresponds to a viewing ray segment through the tetrahedron, whose length l can be computed from the geometry. The method of Shirley and Tuchman [13] is to evaluate opacity once at the "thick" vertex A. The opacity is zero at the other "thin" vertices on the profile. Bilinear interpolation (linear on triangles) in the hardware rendering pipeline is used to interpolate the color and opacity across the triangle, and composite each triangle over the background.

Assume that the color $c(x)$ and differential opacity per unit length $\tau(x)$ vary linearly across the tetrahedron. Then these quantities can be interpolated across faces or edges to give values τ_{front}, c_{front} and τ_{back}, c_{back} at the front and back of the tetrahedron respectively. Shirley and Tuchman [13] show that the total opacity at A is:

$$\alpha = 1 - e^{-\frac{1}{2}l(\tau_{front} + \tau_{back})} . \qquad (1)$$

They define the color at A as the average of the front and back colors: $(c_{front} + c_{back})/2$.

The linear interpolation of color and opacity causes artifacts, which can reveal the separation of the flow volume into tetrahedral cells. This is because the linear interpolation of the opacity does not accurately model the exponential absorption. The opacity at each point should be separately evaluated by equation (1). This requires a linear interpolation of τ and l separately, and then an exponential per pixel, not commonly available in hardware. We use the texture mapping hardware available in our SGI Onyx™

system for this problem. For the case of constant τ per tetrahedron, as in our flow volume application, we put the quantity 1-exp(-*u*) in a one dimensional texture table, indexed by *u*. The texture coordinate *u* is set to zero at the thin vertices, set to τl at the thick vertex, and then interpolated by the shading hardware before indexing into the texture table. If τ is not constant, we put 1-exp($-\tau l$) in a 2D texture table indexed by texture coordinates τ and *l*.

The compositing scheme of [1] and [13] requires that the volume cells be composited in back to front order. In general, sorting for the back to front order is a difficult problem. In the current interactive system, we can avoid sorting by assuming the color of the smoke is uniform, a reasonable assumption for the visual effect we desire. If the color of the smoke is constant, then the final color at each pixel is simply $Color = \left(\prod t_j\right) Color_{background} + (1 - \prod t_j) Color_{smoke}$, which does not depend on the order of compositing. A detailed proof of this is given in [1].

Puffs and Compressible Flows

Controlling the opacity τ of the smoke allows for some interesting effects. By making the opacity depend on the time step in the advection, we can create smoke puffs. The puffs blow along in real time, as long as no other parameters change, bunching up where the current is slower. For compressible flows, we make the differential opacity τ inversely proportional to the volume of the tetrahedron. Here, τ is still constant within each individual tetrahedron and a 1D texture suffices.

Colored Smoke

For scientific visualization purposes, it is useful to represent additional information about the smoke, or the region it passes through (Figure 7). We can allow the color of the smoke to vary, according to some other scalar field, at the expense of sorting the volume elements. We use a general finite-element sort to accomplish this.

This leads to a case where the color also varies linearly across the tetrahedron. The Shirley-Tuchman approximation $(c_{front}+c_{back})/2$ for the color of the thick vertex is not precise; it weights the two colors equally. The frontmost color should have a greater weight because the opacity along the ray segment partially hides the rear color. Williams and Max [14] have found an exact analytical formula for the color in this case, which they implemented with the aid of table lookups. However, the supplementary arithmetic required goes far beyond what is practical in hardware at each pixel. As a compromise, we have used the exact analytical form for the color at the thick vertex, and then used the hardware to interpolate the color across each triangle. The colors of the thin vertices come from their original color specifications, and the opacity is determined, as above, from a texture table. This compromise can be implemented in hardware, and gives a fairly smooth variation that seems to move appropriately when a colored volume rotates.

Flow Volumes in Non-Uniform Fields

The advection of the streamlines used to build the flow volumes assumes a continuous or easily sampled vector field. Scientific data sets are not always computed on regular grids. There are four main grid types we expect to handle: regular, rectilinear, curvilinear, and unstructured. Regular grids are handled with a straightforward trilinear interpolation and either Euler or Runga-Kutta integration. This is the base of our vector field hierarchy, and other data set types or advection integrations are easily added using C++ subclasses. There are two kinds of curvilinear data sets: those specified by a global analytical function, and those specified by arrays of coordinates. A class hierarchy of these various vector fields is outlined in figure 2. Solid bubbles represent class definitions we have already implemented. Hollow bubbles represent class structures we plan to implement or are currently implementing. Atmospheric curvilinear grids describe a field which is regular in two dimensions, and table-based curvilinear in the third (representing altitude which curves to follow the terrain). Unstructured or irregular grids, whose data points follow no pattern and do not have consistent adjacency information, represent another common but more difficult data set.

For curvilinear grids, (Figure 6), the entire vector field is converted from physical to computational space by multiplying each vector by the inverse of the local Jacobian matrix. Now all calculations may be done in computational space, which is regular and easy to work with. The world coordinates in which the smoke cursor exists are in physical space. Thus whenever the cursor is moved, the correct computational space coordinates must be found for the base polygon. This is a two step iterative process. First we walk closer along grid lines to within a reasonably distance then apply Newton iteration to get a more accurate estimate of the position. After the flow is computed, then all its vertices are transformed back to physical space for display. Note the forward transformation is trivial, involving only interpolation, while the inverse is complicated. This method requires five inverse transformations: one for the center, and one for each corner of the initial polygon (which is perpendicular to field lines). At first thought it appears that one could compute the initial polygon with only one inverse transform, then compute the other four points totally in computational space. However, this will lead to severe warping of the polygon when it is later transformed back to physical space.

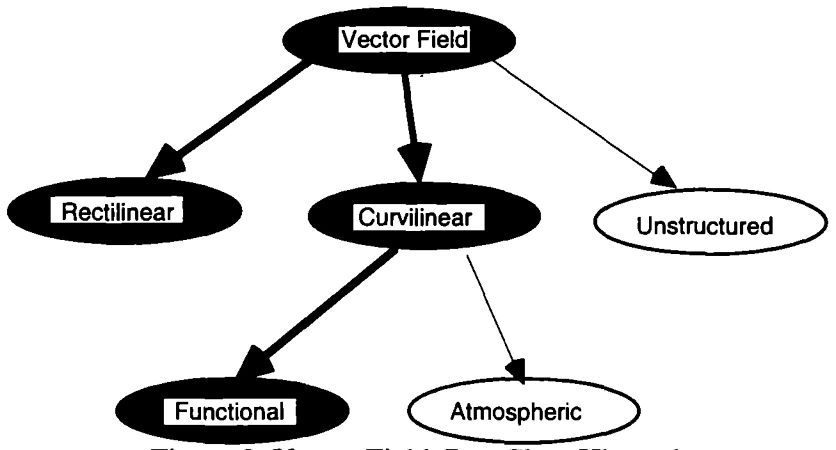


Figure 2. Vector Field C++ Class Hierarchy

The User Interface

Volume rendering is often slow because of the huge number of cells in a typical volume. However, in rendering a flow volume of smoke, only the cells in the small flow volume need be rendered. Everything else is completely transparent, and may be skipped. This makes interactivity possible, and here we describe various interactive controls.

The cursor consists of a jack manipulator customized from SGI's Inventor package. It is attached to a polygon which is always kept perpendicular to the vector field. There are six scale knobs, 2 for each major axis through the polygon, that when selected, will scale the size of the cursor. When the user clicks on the cursor (but not on a scale knob) three orthogonal translation axes appear. Using the mouse to move the cursor in either direction along one of these axis allows for easy translation in 3D. The cursor may move anywhere within the domain of the vector field. If a user tries to move beyond those limits the cursor is constrained to the border. Computing and rendering the flow volume is fast enough for smoke to be drawn continuously while the cursor is moving or the scene is rotating.

Active along with the cursor is an editing window containing sliders for controlling the length of the time step, the number of time steps used, and the smoke's transparency. A color wheel is used to control the smoke color. Toggle buttons are also available to turn the use of transparency texture mapping on or off.

This is useful for machines where texture mapping is available only in software. Toggles are also available for specifying compressible/incompressible flows, puffs, or a time-delayed growing smoke animation. The Inventor SceneViewer already provides menu options for various drawing styles such as dithered or blended transparency, picking styles, and facilities for customized lighting. All viewing transformations like translation, rotation and zooming are handled by a variety of convenient widgets within the SceneViewer.

Texture Splats

The flow volumes allows us to study or probe specific regions of interest within a vector fields. Texture splats allows us to get a more global, but qualitative, view of the vector field. This is accomplished by starting with a volume rendering of an area and adding fine grain detail that is directional. To accomplish this we again relied on the texture mapping hardware of our workstations. We developed a new reconstruction splat for scalar fields as a foundation for this.

Ideal Reconstruction Functions

Laur and Hanrahan [15] also used the fast compositing hardware of the SGI, but approximated each splat by a collection of polygons. Mach bands are visible at the polygon edges, and individual splats are visible, because they do not overlap smoothly. Each splat is typically created from fifteen to twenty-one triangles, or a triangle mesh. For architectures that have hardware support for texture mapping, we can replace these many polygons with a single texture mapped square. Max [16] developed an optimal piecewise quadratic reconstruction function for images. We used this kind of function rather than a gaussian, since its overlap extent is well known and the function goes to zero in a minimum extent. This function was designed for the reconstruction of 2D signals (images), not 3D signals. By focusing on the reconstruction of three-dimensional signals, we have developed a reconstruction function for 3D splats that is accurate from all viewing directions, as described below. This function is a piecewise cubic, offering additional degrees of freedom in the optimization. Hence it is as accurate as [16] for orthogonal views perpendicular to the axes.

We have mathematically optimized the splats to give a smooth overlap, from any viewing direction, with the desired property of minimal extent [2]. If *h(x,y,z)* is the 3D reconstruction filter kernel for a voxel at (0,0,0), its 2D footprint *f(x,y)* is given by:

$$f(x,y) = \int_{-\infty}^{\infty} h(x,y,z)dz. \qquad (2)$$

As in [17], we restrict ourselves to rotationally symmetric filter kernels, such that $h(x,y,z) = g(\sqrt{x^2+y^2+z^2})$, for a function $g(r)$ of a single radius variable r. Our goal is to chose $g(r)$ such that 1) $g(r) \in \mathbf{C}^1$, eliminating mach bands, and 2) the splats overlap into a smooth density, hiding the structure of the individual splats, no matter what the projection direction. To achieve (2), we minimized the relative variance of the 3D reconstruction of a constant function, as the sum of identical splats centered on a 3D integer lattice.

We assume $g(r)$ is zero for $r > t$, and is represented by two cubic polynomials, $p(r)$ for $0 \le r \le s$, and $q(r)$ for $s \le r \le t$. Handling the $\mathbf{C}^1$ constraints by eliminating variables yields an unconstrained optimization problem. There is no absolute minimum in this problem, since the relative variance can be arbitrarily small if t is arbitrarily large. Therefore, we searched for a local minimum with a reasonably small t, and found one at $t = 1.556228$, and $s = 0.889392$. The relative variance of the 3D reconstruction was 0.00119, and the maximum relative deviations from the mean were -0.00233 at $x = y = z = 0.26$, and 0.00534, at $x = y = 0.5$ and $z = 0$. The reconstructed function we get is:

$$g(r) = \begin{cases} 0.557526 - 1.157743r^2 + 0.671033r^3 & 0 \le r \le s \\ 0.067599(t-r)^2 + 0.282474(t-r)^3 & s \le r \le t \\ 0 & s \ge t \end{cases}$$

The integral (2) can be computed in closed form for fixed *(x,y),* since

$$f(x,y) = \int_{-\infty}^{\infty} g(\sqrt{x^2+y^2+z^2})dz = \int_{-\infty}^{\infty} g(\sqrt{r^2+z^2})dz$$

where $r = \sqrt{x^2+y^2}$. (Polynomials of low degree in $\sqrt{r^2+z^2}$ appear in tables of indefinite integrals.) This closed form solution was used to compute the footprint function *f(x,y).* We use this function to generate a texture that is used as the splat. The texture is generated with an extent of 1.6 and the splats are built up in back to front order.

Anisotropic Textures

We can integrate vector fields into the scalar reconstruction function, by adding a slight disturbance in the function, such as tiny vector particles, scratch marks, or a long thin crack. For the master splat, these vector indications are created in the x-axis direction. The scalar reconstruction is still kept at an extent of 1.6, while the vector field generation uses an extent of 2.0. A larger overlap of the vector splats is desired, to produce a texture without gaps. The vector component is windowed separately to produce a smooth texture without hard edges. This allows us to faithfully reconstruct the scalar field while generating a seamless anisotropic texture for the vector field. Figure 3. illustrates a series of twenty such splats. These twenty are part of a larger table that will be explained later in this paper.

Orientation

The primary premise behind splatting is to orient the splats perpendicular to the viewing direction, simulating the integration along the viewing direction. For vector splats, two additional calculations must be carried out. First, the vector field direction for each splat is determined and transformed to viewing coordinates. The xy-projection, (v_x,v_y), of this vector is used to determine a rotation matrix for the splat. The splat is then drawn perpendicular to the viewing direction, but rotated to line up the texture's anisotropic direction with the projected vector field.

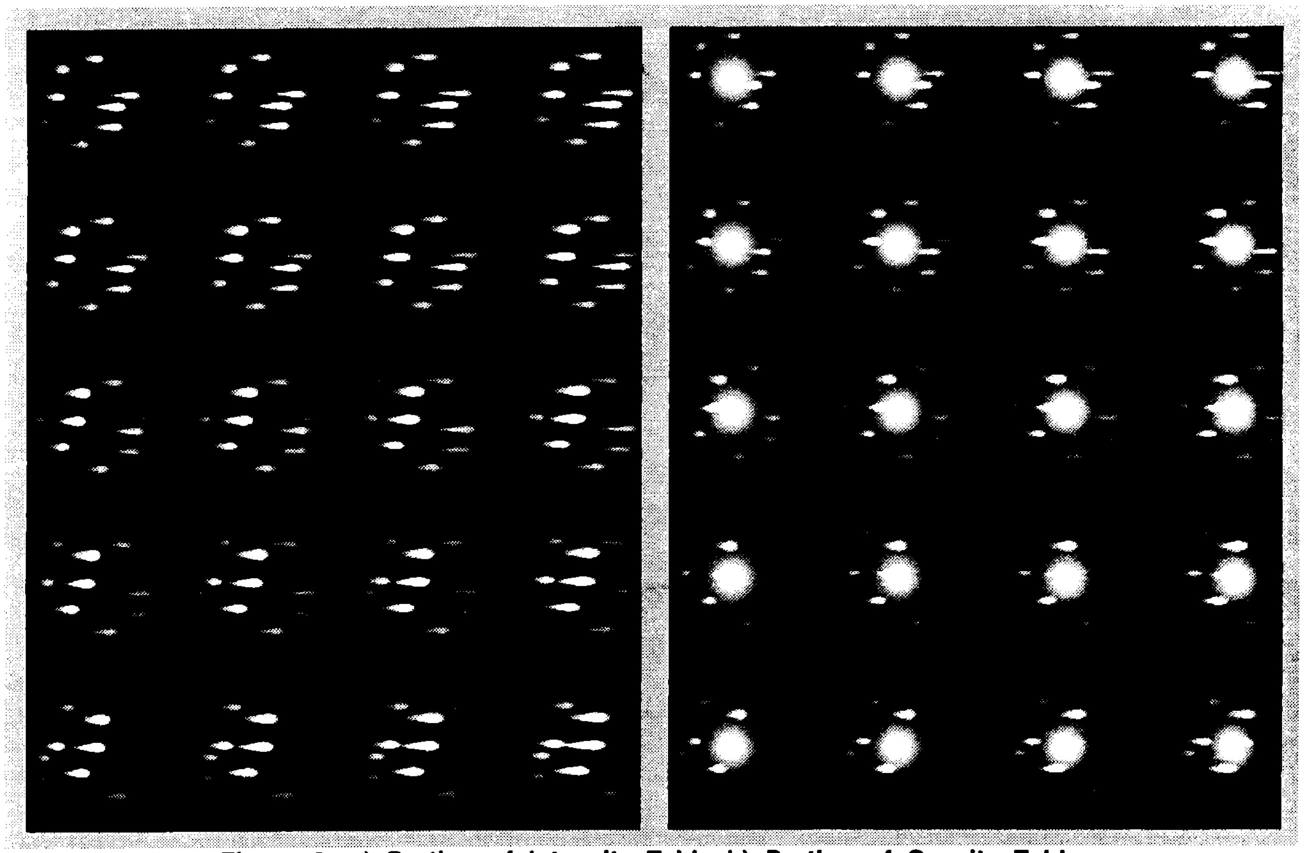


Figure 3: a) Portion of Intensity Table b) Portion of Opacity Table

Compositing

The splat is rendered by selecting an entry from the table in Figure 3a) as an intensity map, and for scalar fields, the corresponding table entry in Figure 3b) for an opacity map. There are several texture mapping possibilities with OpenGL [18]. For our texture splats, we chose a two-component texture consisting of an intensity component I_{tex} and an opacity component A_{tex}. Different texture mapping operations are used for representing only a vector field versus a scalar and vector field.

For a vector field, only, we used the MODULATE operator [18], which multiplies the polygon's color by the splat's intensity and opacity. This operator is used for representing a single vector field. The splats in Figure 3a are used for both the intensity and opacity. The resulting splat's color will thus be:

$$R = R_{in} * I_{tex}$$
$$G = G_{in} * I_{tex}$$
$$B = B_{in} * I_{tex}$$
$$A = A_{in} * A_{tex}.$$

The polygon's color (R_{in},G_{in},B_{in}) (the resulting splat's color) can be used to convey the magnitude of the vector field.

Alternatively, we can map a separate scalar field or use an axis coordinate as an additional spatial cue. The BLEND operator [18] will produce a dissolve between the polygon's color (R_{in},G_{in},B_{in}) and a specified constant color using the splat's intensity (I_{tex}) as the fraction to take from each. The polygon's opacity (A_{in}) is modified as with the MODULATE operator. The resulting splat's color will thus be:

$$R = R_{in} * (1. - I_{tex}) + R_{const} * I_{tex}$$
$$G = G_{in} * (1. - I_{tex}) + G_{const} * I_{tex}$$

$B = B_{in} * (1. - I_{tex}) + B_{const} * I_{tex}$
$A = A_{in} * A_{tex}$

Using the texture maps in Figure 3, this function will give us (R_{const},G_{const},B_{const}) colored vectors, with the appropriately colored scalar volume splats, both of which are attenuated by the polygon's opacity A_{in}. The vector color can be changed for each splat, allowing the vectors to be color coded by magnitude, or offering an additional three-dimensional cue by tying the color mapping to the splat's world coordinate position.

These operations define the texture mapping used to create a data dependent splat with the proper transparency and colors. This splat is what is then composited into the final image. When representing a single vector field, we can stretch the polygon in the vector direction, producing a "streaky" or paint brush affect.

Foreshortening

So far, we have only indicated the xy-projection of the vector direction. No indication is given of the component of the vector directed towards the eye. This can be represented by a foreshortening of the vector based on the viewing direction component in relation to the overall vector length. If we are only representing a vector field or if we separate the vector and scalar splats, an easy method of achieving this is to simply shorten the polygon in the vector direction. This is simply the x-axis direction of the base splat, since we use the transformation pipeline to orient the splat in the direction of the vector. Combined scalar and vector splats are not possible here, since the shortening would compromise the smooth reconstruction of the scalar field. A second alternative is to change the texture mapping coordinates in the x-axis direction (increasing the frequency content of the resulting image). It should be noted that the vector component is windowed in the texture map, with a soft edged mask to produce smooth overlaps, but given a slightly larger extent than the scalar splat. Unfortunately there is no way to automatically have the resulting repetitive texture windowed at the appropriate size using current hardware on the vector texture. A third method, which will also work for combined scalar and vector fields, is to create a table of textures indicating different amounts of foreshortening. This is represented across the columns in Figure 3. The z component v_z of the vector direction is used to index into a column in Figure 3.

Animation

The series of splats represented in Figure 3 also are used to provide animation of the flow field. Going up the rows of Figure 3, the vector component of the splats moves across the scalar reconstruction. We assign each splat a random index into the rows. For animated flows, a changing phase shift is added to the indexing, cycling through the rows and causing the vector texture to move in the direction of the flow. We can vary the speed based on the vector magnitude by controlling the amount of phase shifting.

Conclusions

We have applied these techniques to several application areas. Figure 4 shows a flow volume generated in a turbulent fluid flow study. Aerogels - solid foams that are of such low density that they are almost entirely empty space - have been simulated in Figure 5, where a flow volume highlights the aerogel's insulating capabilities. Figures 6, 10 and 11 illustrate the flow volume and textured splats techniques on a global climate model simulation. In Figure 6, we have used a curvilinear grid in mapping the flow volume from longitude/ latitude/ altitude to the globe. Figure 7 highlights the ability to color code the flow volume using a different scalar field. Here, a ground water contamination study is being probed. The color of the flow volume represents the soil permeability.

Figure 8 illustrates the vector splatting technique on a dummy test tornado data set. The magnitude of the vector field is used to control the opacity and a noisy color map of browns is used to add some variations. Figure 9 uses the blend operation to add white vectors flowing through the volume rendering of the magnitude of the aerogel's airflow. An isocontour of the aerogel particles is used here, where in Figure 5 individual spheres were used. The jet stream winds over North America are represented by vector splats in Figure 10. The magnitude of the winds is mapped to both opacity and color, and a stretched vector splat is used to provide a wispy appearance. The percent cloudiness scalar field from the global climate simulation is volume rendered in Figure 11. The wind field direction is also represented, but color coded by the altitude to give an added depth cue.

Acknowledgments

The authors are indebted to the many people who provided the data used in the visualization in this paper. Tony Ladd and Elaine Chandler provided the aerogel data. The climate data is courtesy of Jerry Potter, the Program for Climate Model Diagnosis and Intercomparison at Livermore, and the European Centre for Medium-range Weather Forecasts. Jeff Hultquist and Al Globus from NASA Ames provided the curvilinear to regular mapping software. Cliff Stein provided the finite-element sorting code for the colored smoke. Roy Hashimoto and Bob Brown from Silicon Graphics were instrumental in integrating this research into Explorer and Inventor. This work was performed under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48, with specific support from an internal (LDRD) research grant.

References

1. Max, N., B. Becker, and R. Crawfis. *Flow Volumes for Interactive Vector Field Visualization.* in *Visualization '93.* October 1993. Los Alamitos, CA: IEEE Computer Society Press.
2. Crawfis, R. and N. Max. *Texture Splats for 3D Vector and Scalar Field Visualization.* in *Visualization '93.* October 1993. Los Alamitos, CA: IEEE Computer Society Press.
3. Schroeder, W.J., C.R. Volpe, and W.E. Lorensen. *The Stream Polygon: A Technique for 3D Vector Field Visualization.* in *Visualization '91.* October 1991. San Diego, CA: IEEE Computer Society Press.
4. Wijk, J.J.v., *Spot Noise-Texture Synthesis for Data Visualization.* Computer Graphics, July 1991. **25**(4): p. 309-318.
5. Hultquist, J.P.M. *Constructing Stream Surfaces in Steady 3D Vector Fields.* in *Visualization '92.* October 1992. Los Alamitos, CA: IEEE Computer Society Press.
6. Leeuw, W.C.d. and J.J.v. Wijk. *A Probe for Local Flow Field Visualization.* in *Visualization '93.* October 1993. Los Alamitos, CA: IEEE Computer Society Press.
7. Gerlick, G.D. *Moving Iconic Objects in Scientific Visualization.* in *Visualization '90.* October 1990. San Diego, CA: IEEE Computer Society Press.
8. Wijk, J.J.v., *Flow Visualization with Surface Particles.* IEEE Computer Graphics and Applications, July 1993. **13**(4): p. 18-24.
9. Hellman, J.L. and L. Hesselink, *Visualizing Vector Field Topology in Fluid Flows.* IEEE Computer Graphics & Applications, May 1991. **11**(3): p. 36-46.
10. Globus, A., C. Levit, and T. Lasinski. *A Tool for Visualizing the Topology of Three-Dimensional Vector Fields.* in *Visualization '91.* October 1991. San Diego, CA: IEEE Computer Society Press.
11. Crawfis, R. and N. Max. *Direct Volume Visualization of Three-Dimensional Vector Fields.* in *Proceedings of the 1992 Workshop on Volume Visualization.* October 1992. New York: ACM SIGGRAPH.
12. Cabral, B. and C. Leedom. *Imaging Vector Fields Using Line Integral Convolution.* in *Computer Graphics.* August 1993.
13. Shirley, P. and A. Tuchman, *A Polygonal Approximation to Direct Scalar Volume Rendering.* Computer Graphics, November 1990. **24**(5): p. 63-70.
14. Williams, P. and N. Max. *A Volume Density Optical Model.* in *Proceedings of the 1992 Workshop on Volume Visualization.* October 1992. New York: ACM.
15. Laur, D. and P. Hanrahan, *Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering.* Computer Graphics, July 1991. **25**(4): p. 285—288.

16. Max, N., *An Optimal Filter for Image Reconstruction,* in *Graphics Gem II,* J. Arvo, Editor. Academic Press: New York. p. 101-104.

17. Westover, L., *Footprint Evaluation for Volume Rendering.* Computer Graphics, August 1990. **24**(4): p. 367-376.

18. Board, O.A.R., *et al., OpenGL Programming Guide.* Release 1 ed. 1993, Reading, MA: Addison-Wesley. 516.

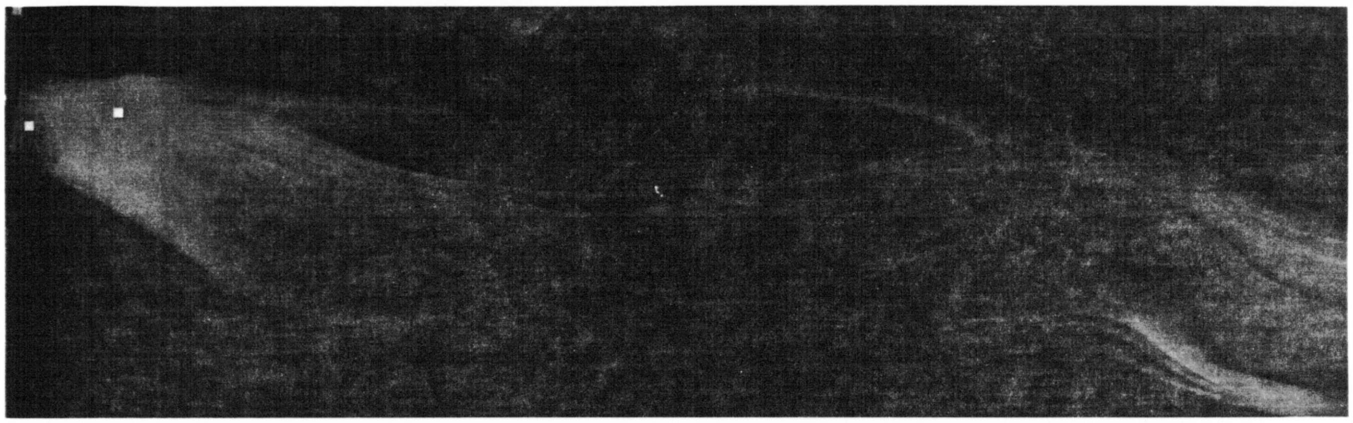
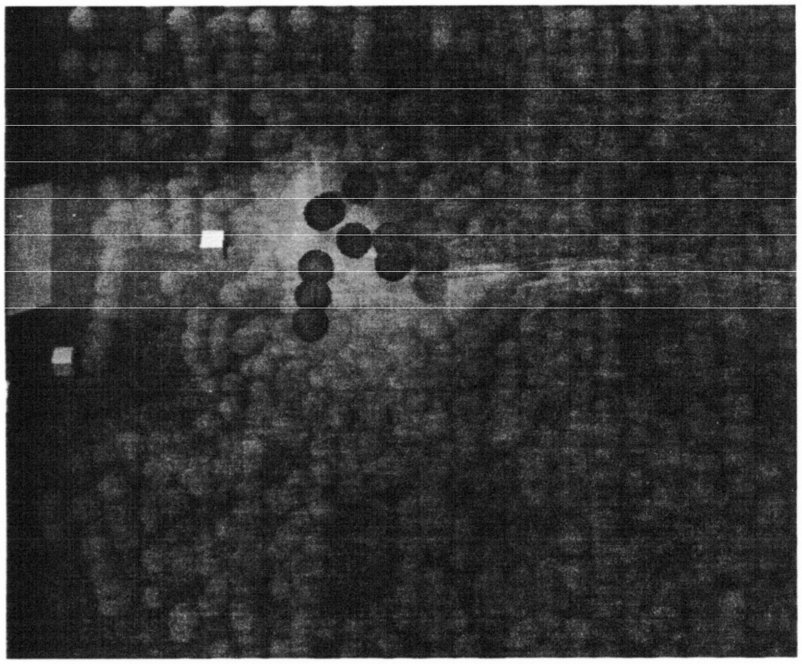


Figure 4. Flow Volume Through a Turbulent Flow



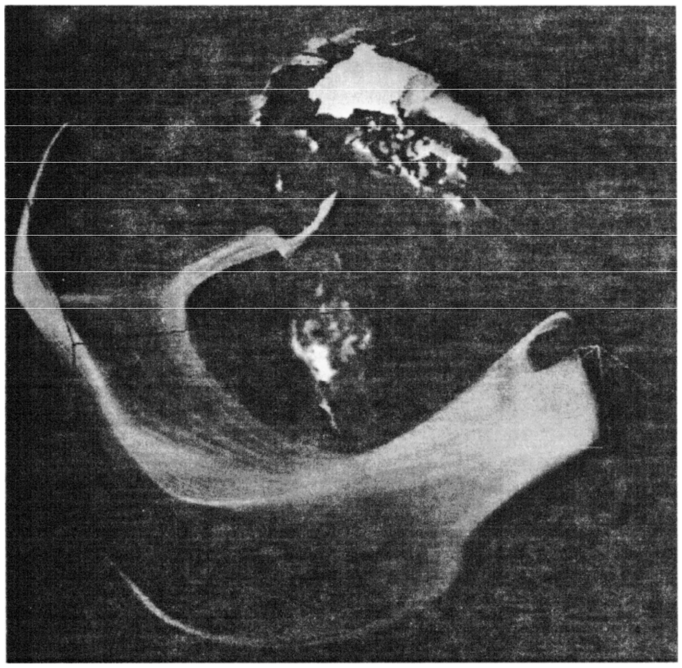


Figure 5. Insulating Aerogel Substrate Simulation.

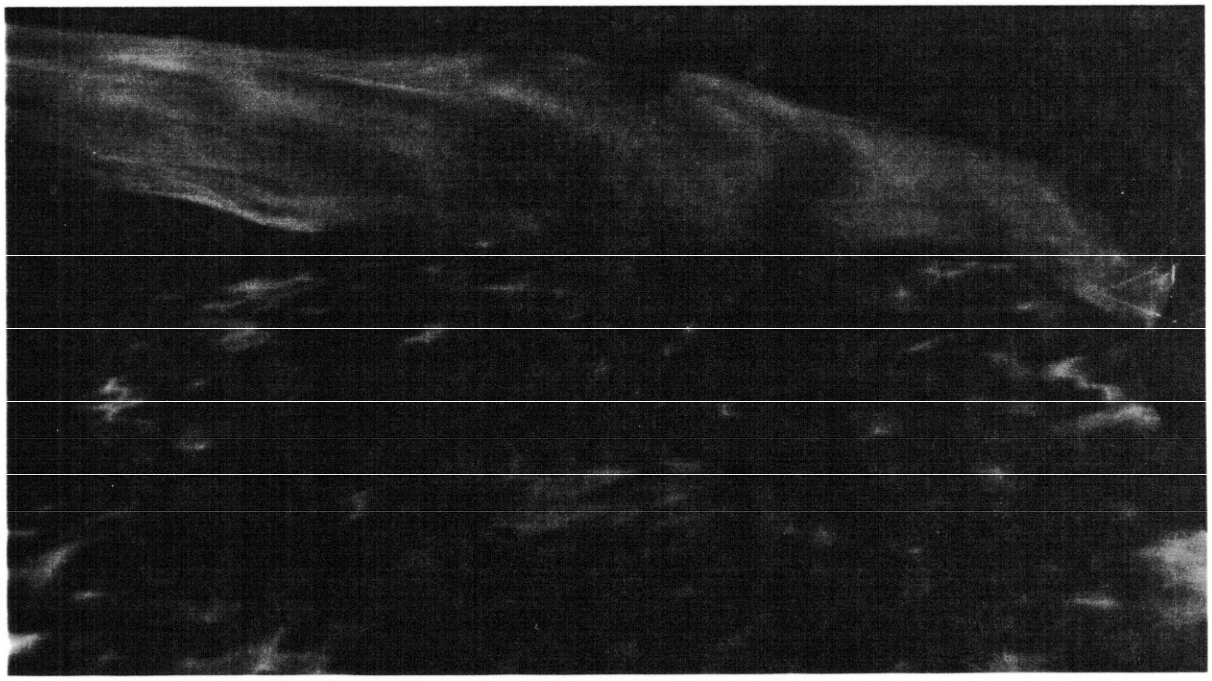


Figure 7. Ground Water Contamination Study colored by Soil Conductivity

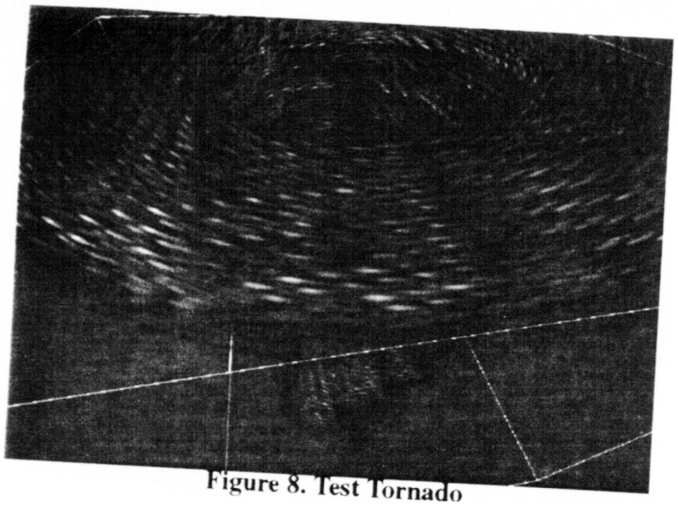


Figure 8. Test Tornado

Figure 9. Airflow through an aerogel substrate. Vector direction in white, magnitude is volume rendered.

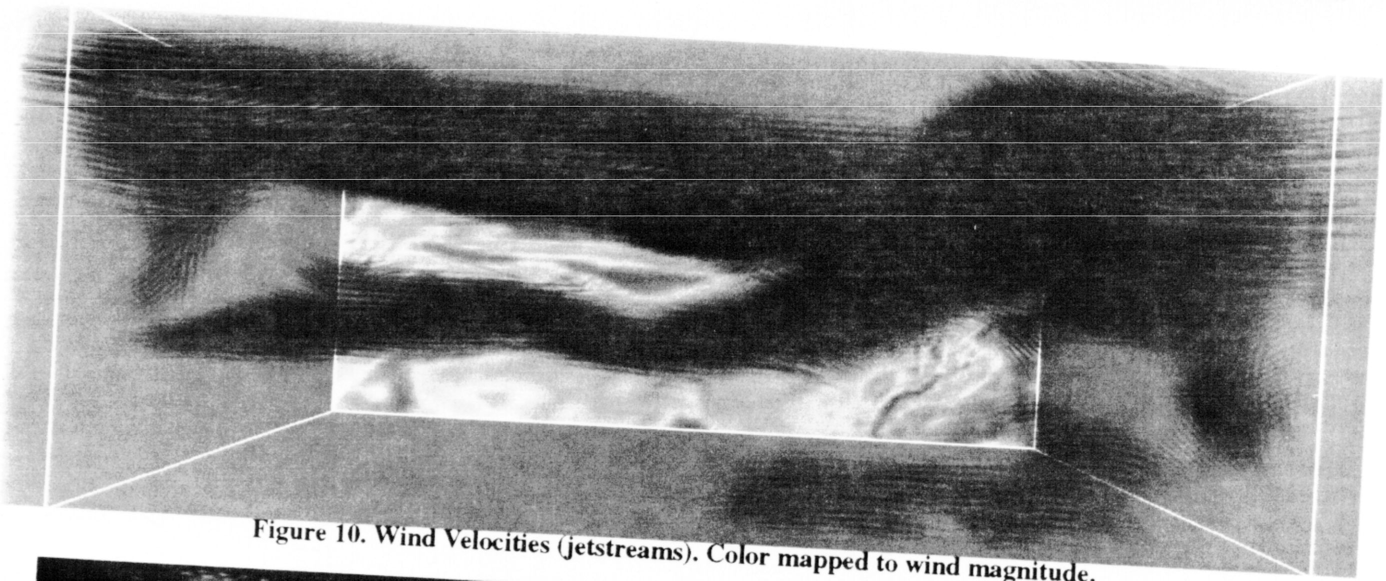


Figure 10. Wind Velocities (jetstreams). Color mapped to wind magnitude.

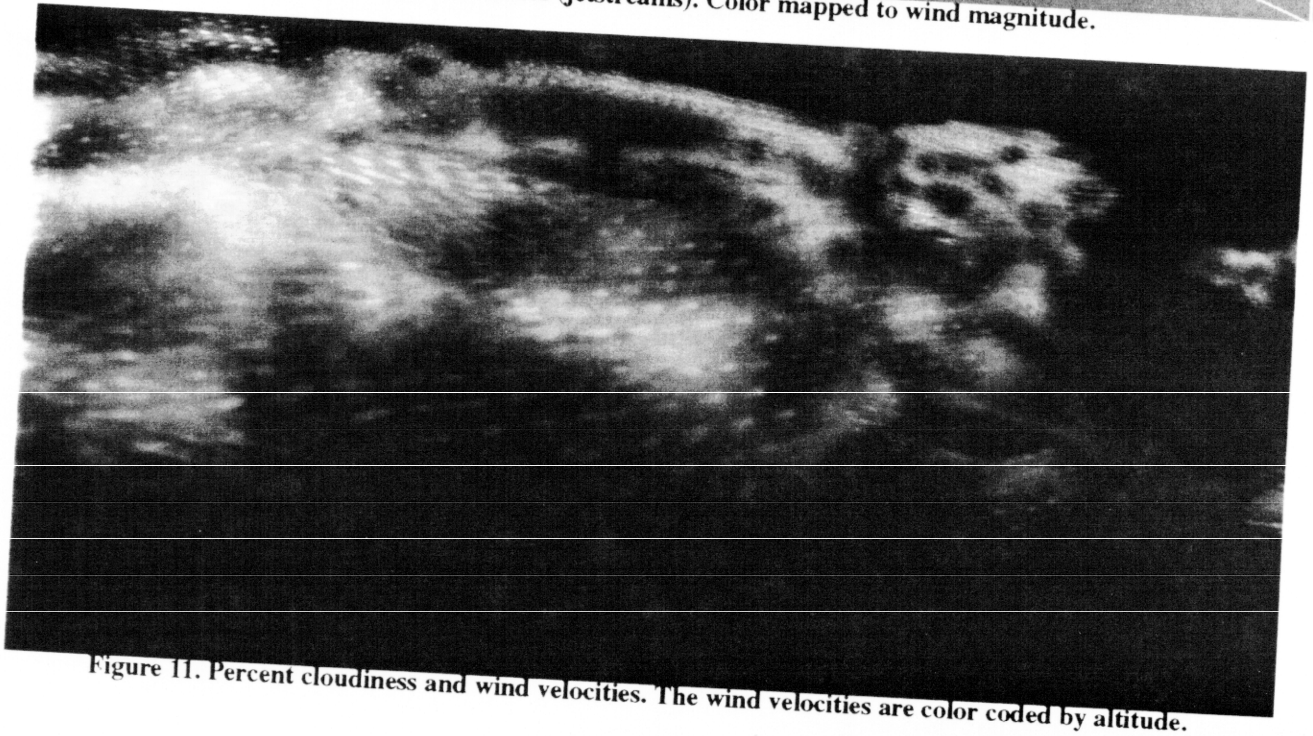


Figure 11. Percent cloudiness and wind velocities. The wind velocities are color coded by altitude.