Vector Plots for Irregular Grids

Don Dovey
Lawrence Livermore National Laboratory

Abstract

A standard method for visualizing vector fields consists of drawing many small "glyphs" to represent the field. This paper extends the technique from regular to curvilinear and unstructured grids. In order to achieve a uniform density of vector glyphs on nonuniformly spaced grids, the paper describes two approaches to resampling the grid data. One of the methods, an element-based resampling, can be used to visualize vector fields at arbitrary surfaces within three-dimensional grids.

1 Introduction

Numerous algorithms for visualizing vector fields have been published. One well-known method, sometimes called an "arrow plot," draws many small line segments or "glyphs" to represent the field. The algorithm is relatively inexpensive and can be used with both 2D and 3D data.

The "arrow plot" algorithm is usually applied to curvilinear and unstructured grids by drawing the glyphs at the grid nodes or at the element centers. There are several disadvantages to this approach. One drawback is that more glyphs are placed in areas where the elements are small than in areas where the elements are large. This variation in glyph density is unrelated to the vector values themselves. A second drawback is that regularity in the grid causes distracting patterns in the output image. Lastly, the user has no control over the glyph spacing in this scheme. In order to alleviate these problems, it is necessary to resample the grid data. Two approaches to resampling the grid are described and compared in this paper.

In the next section, related previous work in the area of vector visualization is reviewed. Section 3 summarizes the algorithm for regular grids. Sections 4 through 7 extend the algorithm to unstructured grids. Some practical techniques for interactively rendering the vector glyphs are described in section 8. Examples and conclusions are discussed in the last two sections.

2 Related work

A set of visualization techniques for flow fields is based on particle advection. Examples include particle paths, particle streams, flow ribbons and flow volumes. These techniques work well for velocity fields but are not necessarily applicable to other vector fields. In electromagnetic wave data, for example, there are periodically-spaced regions at which the magnitude of the electric and magnetic fields is zero — so the data doesn't lend itself to long flow curves. Another consideration with advection-based techniques is that in order to gain a global picture of a flow field, one may need to advect many particles through a volume. This incurs a substantial computational expense and is difficult to achieve at interactive rates.

Crawfis and Max [3] describe a method for direct volume visualization of vector glyphs (drawn as small line segments) which allows them to be displayed in combination with a scalar field on regular grids. Since our method is derived from theirs, we will return to this paper later.

In [4], Crawfis and Max modify their technique for volume visualization of vector and scalar fields by rendering textured splats to the screen. The splatting algorithm makes use of hardware support for polygon texture-mapping in order to achieve interactive performance. Splatting-based techniques and other methods for rendering velocity fields near contour surfaces are discussed in [9]. All of these methods were developed for regular grids.

An algorithm by van Wijk [13] shows flow direction on a parametric surface by rendering an oriented texture on the surface.

Cabral and Leedom [2] developed a "Line Integral Convolution" (LIC) algorithm for textured display of flow fields. The method generates an oriented texture by convolving a local streamline with random noise at each cell in a 2D regular grid. The LIC algorithm was extended to surfaces of curvilinear grids by Forssell [5]. Some weaknesses of the algorithm are that it only shows the component of the vector field that lies in the surface being textured and that it has not been

extended to unstructured grids.

3 Regular grids

The algorithm for regular grids is reviewed in this section, and extended to unstructured grids in later sections. Following [3], we overlay the computational grid with a second regular grid: the *vector grid*. The algorithm selects a random point in each cell of the vector grid and linearly interpolates the vector field values from the computational grid to the selected point. An oriented, anti-aliased line segment centered about the point is then drawn. We refer to the points at which the glyphs are drawn as *vector grid points*. Essentially, the algorithm samples the data on a jittered regular grid. Jittering is required in order to avoid aliasing that occurs with a regular sampling grid [3]. The user selects the spacing of the vector grid and also controls the absolute length of the glyphs.

If the vector field is three-dimensional, the rendering order is important. Back-to-front rendering is necessary because of the way most hardware implements anti-aliased line drawing: partially covered pixels along a line are blended with the previous values in the display buffer. Since the vector points are generated on a semi-regular grid, drawing the glyphs from back to front (or approximately back to front) is trivial. Of course, there is no guarantee that a line segment with its center behind another won't extend in front of the second line segment, but occurence of this case tends to be minimized by the coherence of the field.

4 Finite elements and interpolation

Sections 5 and 6 introduce two different methods for displaying vector plots on curvilinear and unstructured grids. Before the methods are introduced, it is helpful to review some concepts from finite element analysis.

In this paper, we restrict ourselves to unstructured grids composed of linear elements. The element types of most interest are hexahedral and tetrahedral volume elements and quadrilateral and triangular thin shell elements.

For isoparametric finite elements, a set of *shape functions* or *interpolation functions* both defines the geometry of an element based on the element nodes and interpolates quantities from the nodes to the interior of the element. Points in the interior of a hexahedral or quadrilateral element are defined by their

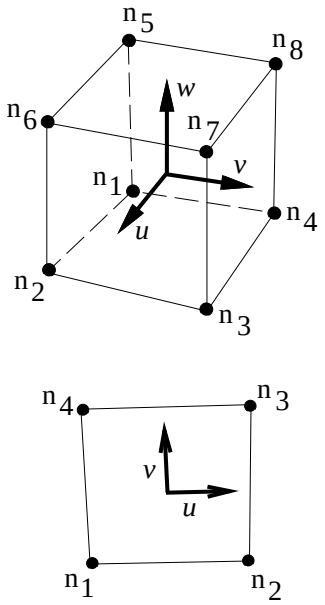


Figure 1: Natural coordinates for a hexahedral and a quadrilateral element.

natural coordinates on the interval (-1, 1) (Figure 1.) For a hexahedral element, interior points are specified by

$$P(u,v,w) = \sum_{i=1}^{8} N_i(u,v,w)P_i \tag{1}$$

$$N_i = \frac{1}{8}(1+uu_i)(1+vv_i)(1+ww_i)$$

Here, (u,v,w) are the natural coordinates of the interior point, N_i are the shape functions, P_i are the global coordinates of the corner nodes, and (u_i,v_i,w_i) are the natural coordinates of corner node i. Similarly, for a 2D or 3D quadrilateral element,

$$P(u,v) = \sum_{i=1}^{4} N_i(u,v)P_i \tag{2}$$

$$N_i = \frac{1}{4}(1+uu_i)(1+vv_i)$$

Linear interpolation is accomplished on triangular and tetrahedral elements with *barycentric coordinates*. For a triangle with vertices P_1, P_2, and P_3, a point P inside the triangle is given by its barycentric coordinates (r,s,t) in the range (0, 1).

$$P(r,s,t) = rP_1 + sP_2 + tP_3 \tag{3}$$

$$r+s+t = 1$$

The barycentric coordinates for a tetrahedron are analogous.

$$P(r,s,t,u) = rP_1 + sP_2 + tP_3 + uP_4 \qquad (4)$$

$$r + s + t + u = 1$$

In either case, the shape functions are simply

$$N_1 = r, N_2 = s, \ldots$$

Note that barycentric coordinates lie on a different interval than natural coordinates.

Interpolating a result value to an arbitrary point in an unstructured grid involves locating the element that contains the point, calculating the natural coordinates or barycentric coordinates of the point with respect to the element, and then interpolating the result value from the element nodes to the point. Interpolation is performed with the shape functions. For an element with n nodes, the result value at the point is given by

$$R = \sum_{i=1}^{n} N_i R_i \qquad (5)$$

R_i in the above equation is the result value at node i.

The reader may refer to a finite element text such as [14] for more information on this topic.

5 Physical space resampling

A natural solution to the problem of varying grid density is to resample the vector field data on a regular or jittered regular grid. The main issue that must be addressed with this method is the cost of initializing the vector grid.

The most expensive step in the procedure for interpolating a result value to an arbitrary point is locating the element that contains the point. This requires a search through the grid elements until the enclosing element is found. The search can be made more efficient with techniques such as hierarchical bounding boxes or spatial partitioning or by making use of spatial coherence. Even with one of these acceleration methods, the search is expensive when repeated for thousands of vector points in a large grid.

Our implementation utilizes hierarchical bounding boxes to accelerate the point tests. At startup, the grid is partitioned into groups of adjacent elements. (Partitioning algorithms are surveyed in [6].) The candidate point is tested against the bounding box for a group of elements before being tested against the bounding box of each element in the group. If both tests pass, then an iterative procedure determines whether the point actually lies inside the element. For efficiency, all bounding boxes are cached in advance. Using hierarchical bounding boxes, we find that it takes from several seconds to several minutes to solve the "point in element" problem for all vector grid points within a 3D unstructured grid.

The discussion that follows assumes a static computational grid. To support interactive animation of transient vector fields, we process the vector grid once during an initialization step and calculate, for each vector point, the element that contains the point and the natural coordinates of the point on that element. This information is stored and used to update the result values at the vector points as the animation steps through time. The sequence of steps for rendering a vector glyph is:

1. Interpolate the vector field to the vector point (Equation 5.)
2. Calculate the physical coordinates of the vector point (Equations 1-4.)
3. Draw the oriented glyph at the point.

The vector glyphs are drawn from back to front by traversing the vector grid in the proper order.

6 Parameter space resampling

In an effort to avoid the expense of initializing the vector grid points, parameter space resampling was developed. The idea is to generate random points directly in the natural coordinates of the elements using an area-weighted (or volume-weighted) distribution. The natural coordinates are then mapped to physical coordinates in the grid. This method relies on the assumption that, with a relatively dense resampling, the precise distribution of the individual points is not critical as long as the overall density is correct. The key to the approach is to make sure the vector points are distributed with uniform probability in physical space.

To begin, consider a 2D grid composed of quadrilateral or triangular elements. The user selects a vector grid density, D, which is the number of vector glyphs to be drawn per unit area. The rendering algorithm visits each element in turn. The number of vector points to be generated for the element is calculated by multiplying the area of the element, A_i, by the vector density. We use a random number generator to decide whether a vector point will be generated for a

particular element as a result of the fractional part of the vector count.

The number of vector glyphs to be drawn in an element is given by N_{vec} in the pseudo-code below.

```
N_vec = floor( D · A_i )
if ( random() <= mod(D · A_i, 1) )
     N_vec = N_vec + 1
```

The `floor()` function rounds a number down to the closest integer, `mod()` returns the remainder of the first argument divided by the second, and `random()` generates uniformly distributed pseudo-random numbers in the range (0, 1).

Once the number of vector points for a quadrilateral element has been calculated, the natural coordinates of each point are randomly generated in the range (-1, 1) by two calls to `random()`. The vector field is interpolated to each vector point from the element nodes (Equation 5) and the physical coordinates of the points are computed (Equation 2.) We may then render the vector glyphs at the points.

This approach deposits points in a quadrilateral element with uniform probability as long as opposite sides of the element are nearly parallel. If an element has a trapezoidal shape, one can obtain a uniform probability for the vector points by splitting it into two triangles and then using the method for triangles.

The method for obtaining a proper distribution of vector points within a triangular element is based on barycentric coordinates. (See [11] for an alternate explanation.) Recall that a point P inside a triangle is given by its barycentric coordinates (r, s, t), where

$$P(r, s, t) = rP_1 + sP_2 + tP_3$$

Holding the value of r constant yields a line segment parallel to the triangle edge $\overline{P_2P_3}$ (Figure 2.)

To find the proper distribution of points on the triangle, we compute the length of a line segment $\overline{P_aP_b}$ which is defined by a constant value of r (Figure 2.) From the definition of barycentric coordinates,

$$\|\overline{P_1P_a}\| = (1 - r)\|\overline{P_1P_2}\|$$

$$\|\overline{P_1P_b}\| = (1 - r)\|\overline{P_1P_3}\|$$

By similar triangles,

$$\|\overline{P_aP_b}\| = (1 - r)\|\overline{P_2P_3}\|$$

The line segment defined by $r = 0.5$ is half as long as $\overline{P_2P_3}$. By comparing the area of thin strips centered about the two line segments and taking the limit as the width of the strips approaches zero, we find that

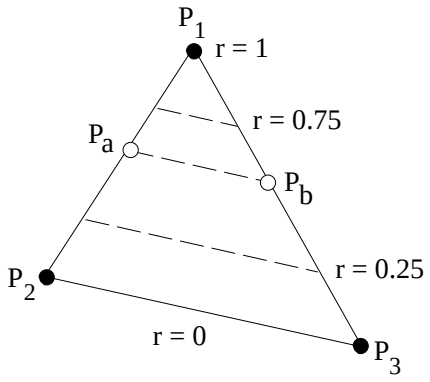


Figure 2: Isoparametric lines in barycentric coordinates.

the probability that $r = 0.5$ is half the probability that $r = 0$. In general,

$$Probability(r) = (1 - r)Probability(0) = (1 - r)C$$

where C is a constant. In other words, the probability distribution is a linear ramp with zero probability at $r = 1$ and maximum probability at $r = 0$. We are interested in the *relative* probability of different values of r, so C is arbitrary.

A random value in the range (0, 1) with the desired linear ramp probability distribution can be generated as follows (see [7].)

```
r = 1 - sqrt(random())
```

A point is equally likely to fall at any position along the line segment defined by a constant value of r, so s and t may be chosen at random with the constraint that $r + s + t = 1$. The constraint implies that s and t must be in the range $(0, 1 - r)$. Random barycentric coordinates may be generated:

```
r = 1 - sqrt(random())
s = (1 - r)*random()
t = 1 - r - s
```

Given these randomly distributed points in 2D, one can draw the vector glyphs interactively as before.

The previous approach extends trivially to surfaces in 3D. For example, it can be used to show a vector field on the exterior faces of a 3D curvilinear grid. The only change is that we need to insure that the vector points are rendered in back-to-front order. One could sort the faces back-to-front [10] before generating and drawing the vector glyphs, but it is simpler and usually more efficient to generate the vector points and sort the points in back-to-front order before rendering.

The technique for triangles enables us to carpet arbitrary surfaces within a 3D grid. For example, surfaces of intersection of a cutting plane with a grid or isocontour surfaces generated by the Marching Cubes algorithm [8] are composed of triangular facets. These surfaces can be generated within regular or unstructured grids. In order to correctly interpolate the vector field from the nodes of the grid to the vector points, we first generate the vector points on a triangular facet and then calculate the natural coordinates of the points with respect to the 3D volume element in which the facet is embedded.

The technique for quadrilateral elements extends trivially to hexahedral elements. The technique for triangles extends to tetrahedra, with one minor difference. If one calculates the area of the triangle defined by a constant value of r in a tetrahedron, one finds that the correct probability distribution for r is $(1-r)^2 C$. Hence, random barycentric coordinates for a tetrahedron are generated as follows.

```
r = 1 - cbrt(random())
s = (1 - r)*(1 - sqrt(random()))
t = (1 - r - s)*random()
u = 1 - r - s - t
```

We omit the proof for brevity, but note that the cube root is needed to generate the desired quadratic probability distribution [7]. As before, points that are generated in a volume grid must be sorted before rendering.

7 Discussion of methods

It's useful to compare the previous two methods in terms of speed and image quality. We consider speed first.

The physical space resampling method requires an expensive initialization, but then the user can modify the view with no additional cost since the sort is trivial. The parameter space resampling eliminates the initialization cost but requires that the vector points be re-sorted each time the view changes. The efficiency of the two methods is dependent on the implementation, but an example should help indicate the general differences.

We generated 16,000 sample points on a small 3D grid containing 14,000 hexahedral elements. Initializing the points for the physical space resampling required 25.2 seconds with hierarchical bounding boxes, and 186.0 seconds without the hierarchical test (but still using bounding boxes for the elements.) Sorting 16,000 points in the parameter space method required 1.4 seconds. In either case, rendering required 12 seconds with antialiasing and 2 seconds with no antialiasing. All times are for an SGI Indigo Elan R3000 workstation.

One can reduce the cost of sorting in the parameter space resampling method by generating the vector points in parameter space and then storing them in a voxel grid. That way, the program only needs to sort the points within each voxel when the view changes. This "hybrid method" may be useful if the sorting time is found to be a bottleneck.

In terms of image quality, there are some differences between the two algorithms. The parameter space resampling method allows the density of glyphs to vary from element to element, as a result of rounding the fractional part of the glyph count. The physical space resampling maintains a constant density from cell to cell. Thus, the parameter space resampling permits more local variation in the vector point distribution, but this tends to be less noticeable as the vector density increases. In general, images generated with the same sampling density by the two methods show some differences in the amount of "clustering" but are not drastically different in quality (Figures 3–5.)

In a Lagrangian analysis setting (i.e. simulations where the nodes of a grid may move as the material deforms), one can either regenerate the vector grid points at each timestep or retain the old vector grid points in parametric coordinates. In the latter case, the vector points may change physical position as the elements are distorted. The point distribution should remain relatively uniform unless individual elements undergo a significant change in volume, which is uncommon in practice. If the old vector points are retained, they should be re-sorted at each new timestep to maintain the back-to-front rendering order.

8 Rendering techniques

The following paragraphs look at some ways to display particular aspects of a vector field and to address difficulties associated with vector plotting.

Vector magnitude can be shown by scaling the lengths of the individual vector glyphs. Another way to show magnitude is by colormapping the glyphs — that is, the hue of each glyph is selected by a lookup in a color table. These two methods can be used simultaneously.

A line segment shows the local orientation of the field, but does not show whether the field is pointed in the positive or negative direction along that line.

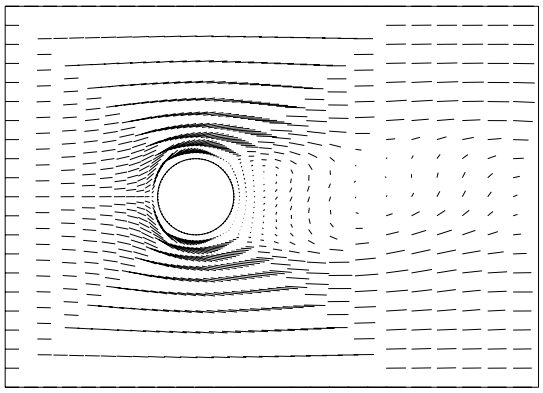


Figure 3: Velocity field with vectors drawn at the grid nodes.

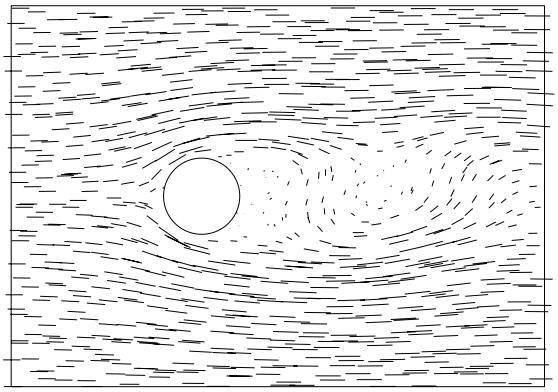


Figure 4: Physical space resampling.

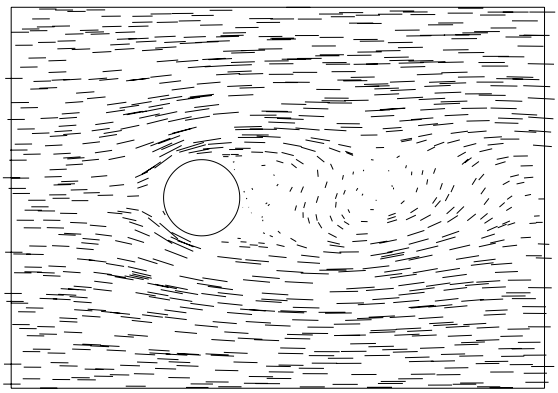


Figure 5: Parameter space resampling (same sampling density as Figure 4.)

The standard technique for showing vector direction is to draw an arrowhead at one end of the line segment. This increases the rendering cost (particularly for 3D vectors) and can clutter a dense image. An alternate way to show vector direction is to vary the hue along the length of the line segment. For example, make the tail end of the vector green and the head end red, with linear interpolation of the two colors in between. If the vector glyphs are being colormapped, one can colormap one end of the glyphs and set the other end to a constant color in order to show direction.

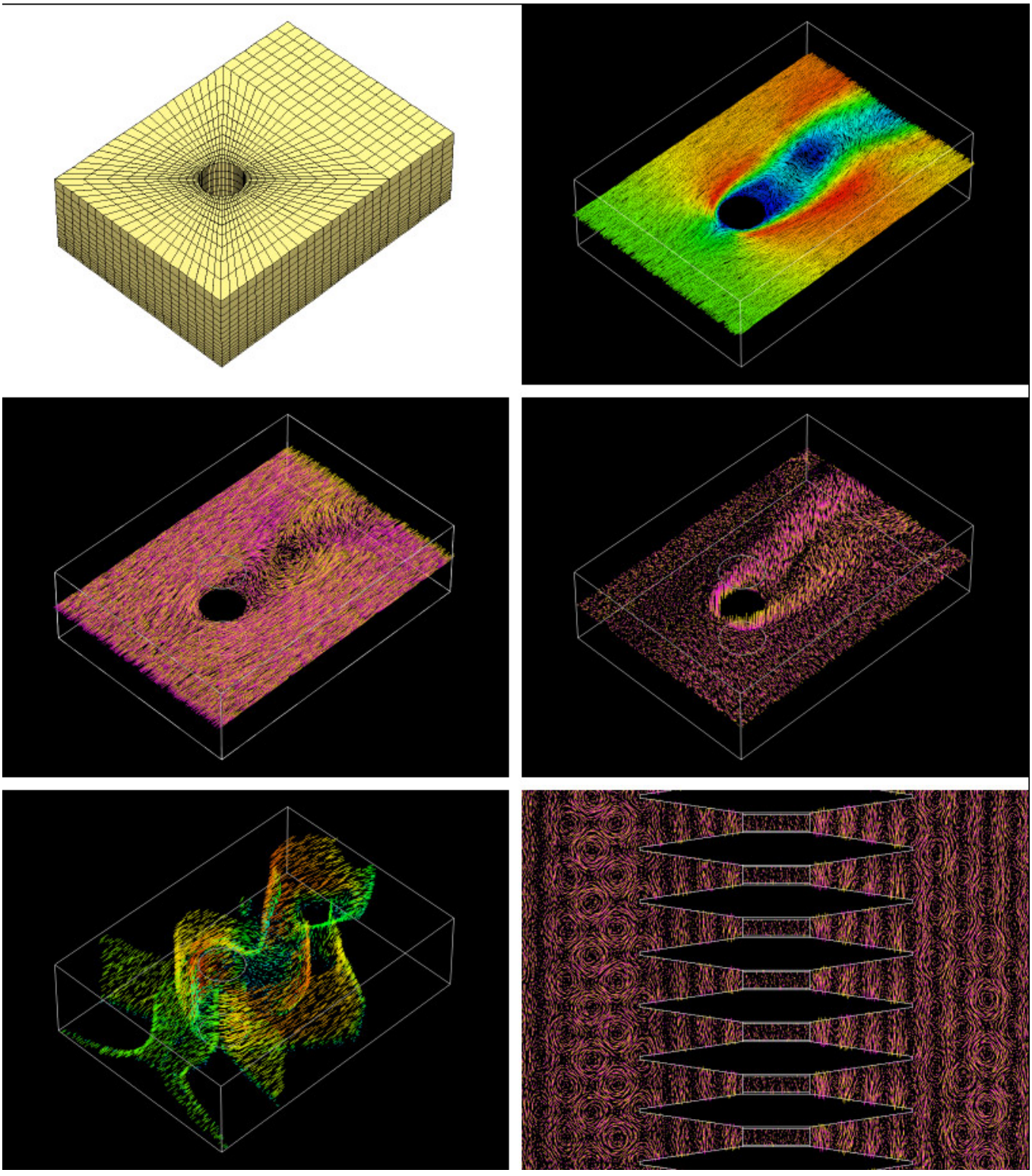
Max [9] points out that the 2D projection of a 3D line segment is ambiguous, since many 3D segments can have the same projection. Hence, it is difficult to see how 3D vector glyphs are oriented with respect to the viewer unless the view is rotated interactively or unless the glyphs themselves are animated. One way to partially overcome this problem is to modulate the brightness over the length of the glyph. If the vector is parallel to the view direction, the brightness of the farthest end from the viewer is scaled to zero (or to some value between zero and one) while the closest end is drawn at full brightness. If the vector is perpendicular to the view direction then both ends are drawn at full brightness. At other angles, the end values are interpolated between these extremes. The purpose of this technique is to give the vector glyphs an exaggerated appearance of perspective scaling. Another way to help show line orientation is with simulated lighting, a subject which other researchers have covered well [1].

If the vector grid is dense then vectors that are close to each other may overlap, making them indistinguishable. This can be addressed by jittering the brightness of the vectors as in [9].

When rendering 3D data, values in the middle or rear of the volume tend to be obscured by values at the front of the volume. Often, the best technique for visualizing the interior of a 3D grid is to display the vector glyphs at a planar slice through the grid. The slicing plane can be moved through the volume to let the user view all parts of the model.

9 Examples

Figures 6 to 10 illustrate a fluid dynamics problem in which an incompressible fluid flows past a cylindrical post and a plate (the plate is at the bottom of the grid.) The grid in this example was treated as unstructured. Figures 7 and 8 show the velocity at a horizontal cut plane. Vortices which form behind the post and are carried downstream by the flow are



visible in the images. The relative magnitude of the velocity field is shown in Figure 7; the direction of flow is emphasized in Figure 8. In Figure 8, the tails of the vector glyphs are magenta and the heads are yellow. Regions of the cut plane that are more yellow or more magenta show areas where the flow direction is partially out of the cut plane. Figure 9 shows vorticity at the cut plane. Vorticity is a vector field which measures the local change in velocity. Figure 10 shows the flow velocity at six isosurfaces of pressure.

A snapshot of the electric field in an electromagnetic wave simulation is shown in Figure 11. The wave travels from left to right in the image, and its energy is deflected downward by the window in the center.

Figures 7 to 10 were generated with parameter space resampling, and Figure 11 was generated with physical space resampling.

10 Conclusion

Vector plots provide a way to interactively visualize vector fields. The technique can be used with 2D and 3D vector data on regular and irregular grids. For unstructured grids, physical space and parameter space resampling methods were described. Parameter space resampling is efficient and allows us to visualize vector fields at arbitrary 3D surfaces. Physical space resampling guarantees that sample points are well-distributed.

Acknowledgements

The author would like to thank Roger Crawfis, Dan Schikore, and the reviewers for their helpful comments on this paper. Datasets in the images were generated by Mark Christon, Scott Nelson and Teresa Swatloski.

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.

References

[1] D. C. Banks, "Illumination in Diverse Codimensions," *Computer Graphics Proceedings '94*, ACM Siggraph, New York (1994) pp. 327-334.

[2] B. Cabral and L. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Computer Graphics Proceedings '93*, ACM Siggraph, New York (1993) pp. 263-270.

[3] R. Crawfis and N. Max, "Direct Volume Visualization of Three Dimensional Vector Fields," *Proceedings, 1992 Workshop on Volume Visualization*, ACM Siggraph, New York (1992) pp. 55-60.

[4] R. Crawfis and N. Max, "Texture Splats for 3D Scalar and Vector Field Visualization," *Proceedings, Visualization '93*, IEEE Computer Society Press, Los Alamitos, CA (1993) pp. 261-266.

[5] L. Forssell, "Visualizing Flow Over Curvilinear Grid Surfaces Using Line Integral Convolution," *Proceedings, Visualization '94*, IEEE Computer Society Press, Los Alamitos, CA (1994) pp. 240-247.

[6] B. Hendrickson and R. Leland, *The Chaco User's Guide*, Sandia National Laboratories, SAND93-2339, Albuquerque, NM (1993).

[7] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA (1969).

[8] W. Lorensen and H. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics Proceedings '87*, ACM Siggraph, New York (1987) pp. 44-50.

[9] N. Max, R. Crawfis and C. Grant, "Visualizing 3D Velocity Fields Near Contour Surfaces," *Proceedings, Visualization '94*, IEEE Computer Society Press, Los Alamitos, CA (1994) pp. 248-255.

[10] M. Newell, R. Newell and T. Sancha, "A Solution to the Hidden Surface Problem", *Proceedings of the ACM National Conference 1972*, pp. 443-450.

[11] G. Turk, "Generating Random Points in Triangles," in Graphics Gems, edited by Andrew Glassner, Academic Press, San Diego, CA (1990).

[12] A. Van Gelder and J. Wilhelms, "Interactive Animated Visualization of Flow Fields," *Proceedings, 1992 Workshop on Volume Visualization*, ACM Siggraph, New York (1992) pp. 47-54.

[13] J. van Wijk, "Spot Noise," *Computer Graphics Proceedings '91*, ACM Siggraph, New York (1991) pp. 309-318.

[14] Zienkiewicz, O. C. and Taylor, R. L., *The Finite Element Method, Fourth Edition, Volume 1*, McGraw-Hill, London (1994).