Comets: A Fast, Intuitive, 3D Vector Glyph

Alejandro Aceves-Gaona and Robert R. Lewis
Washington State University
School of Electrical Engineering and Computer Science
e-mail: {aaceves, bobl}@tricity.wsu.edu

Abstract

We present a new glyph called a "comet" to represent vectors. Comets improve the accuracy of vector visualization to enhance understanding while reducing visual clutter. Also, they eliminate an ambiguity possible with conventional vector glyphs.

By using a hue-lightness-saturation (HLS) color space, devoting saturation to distinguish head from tail, and partially devoting lightness to represent the depth component of a vector's direction, comets allow increased visual accuracy and keep hue and (partially) lightness free to encode other quantities such as the nature of the object to which the vector is attached, if any.

Comets are computationally inexpensive and can represent large data sets. We show how it is possible to use the OpenGL™ lighting model to place comets in static display lists that can be viewed by a moving camera without reconstruction.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms ; I.3.8 [Computer Graphics]: Applications—Displaying Library

Keywords: vector display, scientific visualization

1 Introduction

The primary goal of scientific visualization is to present large quantities of complex data in a way that is easily assimilated by the human visual system. A basic problem in this area is to visualize "fields": physical quantities which are not normally directly visible such as temperature, pressure, velocity, and other scalar, vector, and even higher-rank tensors. Such fields are defined over a domain of a given dimensionality.

The fact that the resulting representation must often be shown on a two-dimensional display complicates the matter even further: One must somehow reduce the intrinsic dimensionality of the data. This representation often involves "glyphs": graphical representations of fields.

In this paper, we will consider the common problem of the representation of three-dimensional vectors embedded in a three-dimensional volume on a two-dimensional display. Vectors may or may not be associated with particles, lines, surfaces, or other entities and both vectors and the other entities may have their own distinct glyphs.

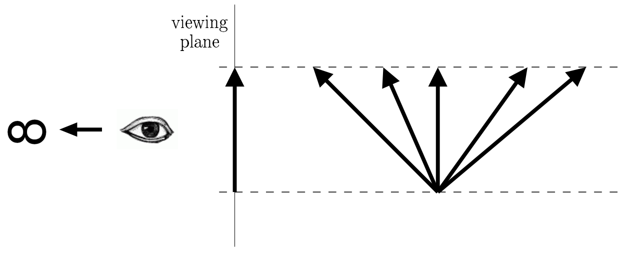


Figure 1: Vector Projection Ambiguity. Any one of the five vectors on the right has the same projection on the viewing plane, making them indistinguishable.

Most vector glyphs are based on the projection onto the display of a line (segment) originating at the vector location: the point in space at which the vector is measured, pointing in the direction of the vector, and scaled by some factor, often under control of the user. As noted by [Max et al. 1994], however, this projection is subject to the obvious projection ambiguity illustrated in Figure 1: The component of the vector along the line of sight cannot be determined.

To improve matters, this line is usually drawn with an arrow on the end opposite the vector location to emphasize the vector direction. If this arrow is drawn properly, it is possible to somewhat distinguish large magnitude vectors pointing towards or away from the viewer from small magnitude vectors perpendicular to the viewer, but whether the vector is going away from or towards the viewer is still indeterminate. Furthermore, it now becomes necessary to draw three or four lines (or, for efficiency, one OpenGL "line strip") instead of one. Sometimes, a wireframe pyramid is set in place of the usual dihedral arrow.

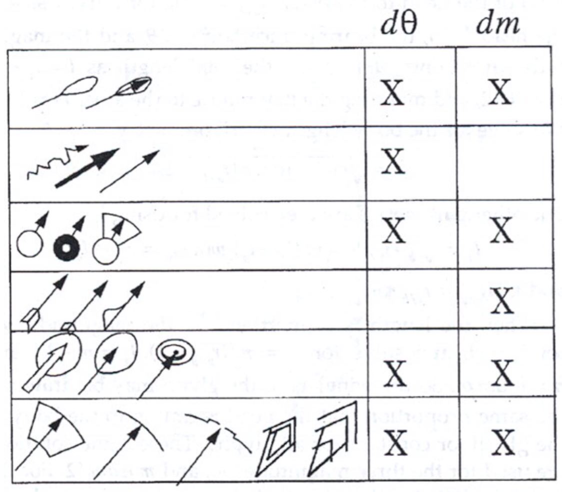
The indeterminacy can be removed by drawing a solid polyhedral cone to represent the arrow head, but rendering efficiency drops even further in order to draw the polyhedral facets. Furthermore, as the glyphs get increasingly complicated, projections of adjacent vector glyphs can overlap and give rise to confusion.

In this paper, we present a new glyph for three-dimensional vectors called "comets" which removes directional and magnitudinal ambiguities at negligible loss of rendering speed and with minimal confusion for adjacent vectors. Section 2 reviews previous work in this area. Section 3 describes comet design. Section 4 covers their implementation in OpenGL. Section 5 shows various vector fields rendered with comets and presents performance measurements. Finally, Section 6 presents our conclusions and proposes future work.

2 Previous Work

A large amount work has been done on the display of vector fields. Although most of it is connected to flow visualization, many of the techniques are more generally applicable.

[Max et al. 1994], which we have already discussed, includes a



	$d\theta$	dm
	X	X
	X	
	X	X
		X
	X	X
	X	X

Figure 2: Vector Glyphs Representing Uncertainty (from [Wittenbrink et al. 1996]). An "X" in the $d\theta$ column denotes glyphs that encode angular uncertainty. An "X" in the dm column denotes glyphs that encode magnitudinal uncertainty.

comparison of 12 field visualization techniques: hedgehogs, particle traces, streamlines, stream ribbons, flow volumes, textured splats, stream surfaces, line integral convolution (LIC), spot noise, line bundles, constrained stream lines, and "hairs". Many of these are available in standard visualization packages such as VTK [Schroeder et al. 1998]. They also note the confusion caused by densely scattered icons overlapping and occluding each other. (We will refer to this as the "crowding problem".) They propose four techniques for visualizing vector fields near surfaces using: 1) motion blur, 2) voxel grid containing integral curves, 3) antialiased lines, and 4) hairs sprouting from the surface bent by the vector field. All of them project vectors onto a surface. Only one proposed method is view-dependent.

[Dovey 1995] points out that the conventional arrow vector glyph aggravates the crowding problem, especially in the case of the irregular grids which are his particular emphasis. His solution is to replace the arrow with a line segment with differing colors for the head and the tail: a hue encoding. He also mentions, but does not show, the use of brightness instead of hue for the same purpose. Both encodings would still be subject to the projection ambiguity.

[Boring and Pang 1996] propose vector glyphs that, using a hue-saturation-value color scheme, encode vector magnitude as line segment hue or value and encode vector direction as lighting conditions and *vice versa*. This minimizes crowding. It reduces, but does not eliminate, the projection ambiguity, as the viewer must understand the lighting geometry and must use multiple lighting geometries to understand all components of the direction in the latter case.

[Wittenbrink et al. 1996] creates a new set of vector glyphs to encode uncertainty in environmental vector fields, as shown in Figure 2. These glyphs decrease the intuitiveness of the display, thereby requiring more viewer training. They also complicate the crowding problem.

3 Design

A vector glyph should be:

- intuitive, requiring a minimum of training for viewers to interpret the vector components.

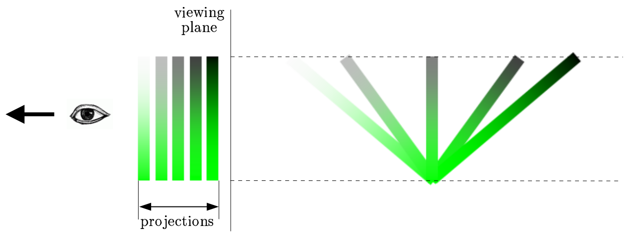


Figure 3: Comet Vector Glyphs. The use of saturation to distinguish vector head from tail and lightness to encode the vector component in the viewing direction prevents the projection ambiguity.

- fast, requiring a minimum of rendering time.
- unambiguous, avoiding projection ambiguity.
- flexible, being applicable to many different applications.
- economical, using as few graphical attributes as possible to encode the vector components.
- simple, so as to minimize the crowding problem.

To satisfy the speed and simplicity criteria, we propose to represent vectors as single line segments whose length encodes vector magnitude, as is done for hedgehogs [Klassen and Harrington 1991], but with additional enhancements.

To the best of our knowledge, no research has used saturation to encode any aspect of a vector, nor has lightness been used to encode vector direction. We propose to do both with glyphs we refer to as "comets", as shown in Figure 3. The head and tail have differing encodings of lightness, saturation, and (at the user's discretion) hue which remove the projection ambiguity.

The head of the vector is completely unsaturated, so its hue is irrelevant. Its lightness encodes the viewer component of the vector direction, increasing monotonically as the vector points more and more towards the viewer. One simple formula for the percent lightness is:

$$L\left(\widehat{\mathbf{U}}\right) = 50\left(\widehat{\mathbf{V}}\cdot\widehat{\mathbf{U}}\right) + 50 \qquad (1)$$

where $\widehat{\mathbf{V}}$ is the normalized viewing direction and $\widehat{\mathbf{U}}$ is the normalized vector being rendered. Note that this maps any vector $\mathbf{U}$ to a lightness percentage in the range $[0,100]$.

There are two ways to color the tail. The first is with full (100%) saturation. (When the saturation is full, the lightness is, by definition, 50%.) We will refer to this as "fully saturated" tail coloring. The user is then free to specify the tail's hue. This guarantees that head and tail will always be distinguishable. This is the method we recommend, especially when comets are the sole indicators of position: i.e., there are no particle, surface, or similar glyphs.

The other way to color the tail is more obvious: allow the user to assign an arbitrary RGB value. We refer to this as "RGB" tail coloring. This would allow the tail to be drawn with an effective lightness or saturation of less than 100% and could lead to an ambiguity about where the head and tail were. It also makes a choice of background color more problematic. On the other hand, it allows comets to work with an existing application that has a preexisting color encoding for other glyphs.

Whether we use fully saturated or RGB tail coloring, the tail color is free to encode application-specific properties such as particle type, mass, charge, pressure, temperature, or (redundantly) magnitude.

The appearance of the pixels drawn between the head and the tail is implementation dependent, but is typically a linear interpolation

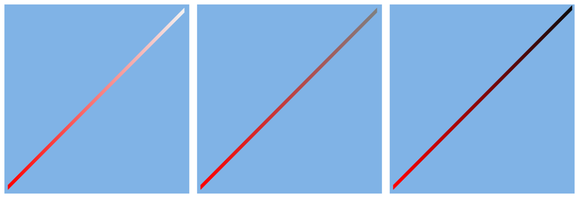


Figure 5: Vectors drawn with the GLUV Vector Utility Library. The component in the viewing direction is directed towards (left), perpendicular to (center), or away from (right) the viewer.

in RGB space. Similarly, for the usually small subset of vectors which project to a single pixel, the appearance depends on implementation details of the line rasterizer[1].

In addition to tail color, all other line segment attributes such as width or stippling ("dashing") are available to encode other properties. The user may also choose to use antialiasing or not depending on appearance vs. performance tradeoffs.

Comets satisfy all of the design criteria indicated above.

4 Implementation

We discuss three aspects of comet implementation: a utility library for vectors, an efficient way to put comets, which are view-sensitive, in display lists, and a useful optimization we found.

4.1 The GLUV Library

We have implemented the design of Section 3 in the form of a small OpenGL utility library similar to GLU[Neider et al. 1993] and GLUT[Kilgard 1996] called "GLUV" which allows the user to treat vectors (i.e. comets) as OpenGL primitives similar to lines and points. Figure 4 shows its application program interface. Note that it allows either fully saturated or RGB tail coloring.

Figure 5 shows some examples. Each vector is drawn with a red tail and has the same component perpendicular to the viewing direction. The only differences are in the component in the viewing direction.

Note that since the head of every comet is some shade of gray, no shade of gray should be used as a background color. We recommend instead the use of a single, light, medium-saturated hue (i.e., a pastel) for a background. We should also remind the reader not to assume that all printer and monitor gamuts are identical and to verify all color combinations empirically.

Because they were generated by software with a preexisting color encoding, the comets in Figures 6 and 7 were generated using RGB tail colors that reflect that encoding. As we mentioned in Section 3, the fact that the atoms (spheres) associated with each vector indicate the tail of the vector, this removes a possible source of confusion. The intuitiveness is maintained.

4.2 Putting Comets in Display Lists

When animating a highly dynamic simulation, each frame generally needs to be reconstructed from scratch. Occasionally, however, it is desirable to look at a single static frame of a simulation from varying viewpoints. In this case, the data itself does not change; only the camera position changes. In such cases, applications can make use of OpenGL "display lists" to re-render data with new viewing parameters. Since comets are view-dependent, this would appear to

[1]This is, in fact, a sampling problem.

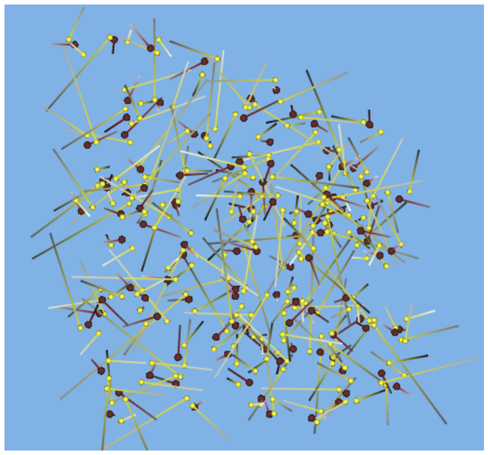


Figure 6: GLUV example in molecular dynamics showing a solvent of a molecule that contains 333 atoms (and vectors)

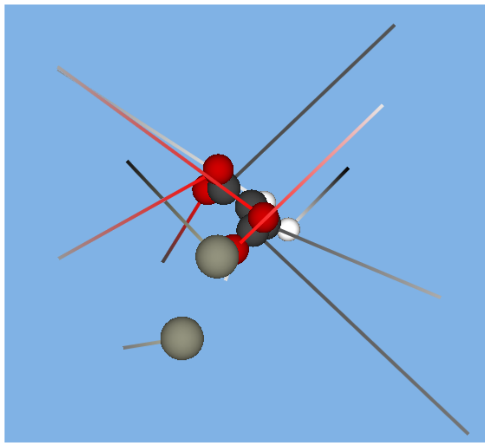


Figure 7: GLUV example in molecular dynamics showing a solute of a molecule.

prevent their being used as part of display lists, but there is a way to permit this.

For a single directional light source impinging on a diffuse surface with an emissive component, the OpenGL lighting model given in [Neider et al. 1993] simplifies to:

$$L = L_e + k_d \max\left(0, \widehat{\mathbf{N}} \cdot \widehat{\mathbf{S}}\right) L_d \tag{2}$$

where L is the resulting color value, L_e is the emissive color (set with *glMaterial*()*), k_d is the diffuse coefficient (also set with *glMaterial*()*), $\widehat{\mathbf{N}}$ is the surface normal (set with *glNormal*()*), $\widehat{\mathbf{S}}$ is the light direction (set with *glLight*()*), and L_d is the incident ("diffuse") light color (also set with *glLight*()*). All quantities are multispectral (RGB) except $\widehat{\mathbf{N}}$ and $\widehat{\mathbf{S}}$.

While this equation is normally used to color the vertices of polygons, we have found on all OpenGL implementations we have studied (see Table 1), that it may also be used to color line vertices. Once set, either directly (with *glColor*()*) or by the lighting model, vertex colors are used to render all OpenGL primitives in the same way. In particular, if smoothing is enabled for lines, the vertex colors, however they are derived, are interpolated in RGB during scan conversion of any primitive[2].

[2]It is possible to attach normals to OpenGL points, but we have yet to

```
void gluvBegin(void);
void gluvEnd(void);

void gluvVector3dv(const double *p, const double *v);
void gluvVector3d(const double  x, const double  y, const double  z,
                  const double vx, const double vy, const double vz);

void gluvTailHued(const double hue);
void gluvTailColord(const double red, const double green, const double blue);

void gluvVectorWidth(const float newWidth);
void gluvVectorSmooth(const int antiAlias);

void gluvEnableLighting(GLenum light_type);
```

Figure 4: Header File for the GLUV Library.

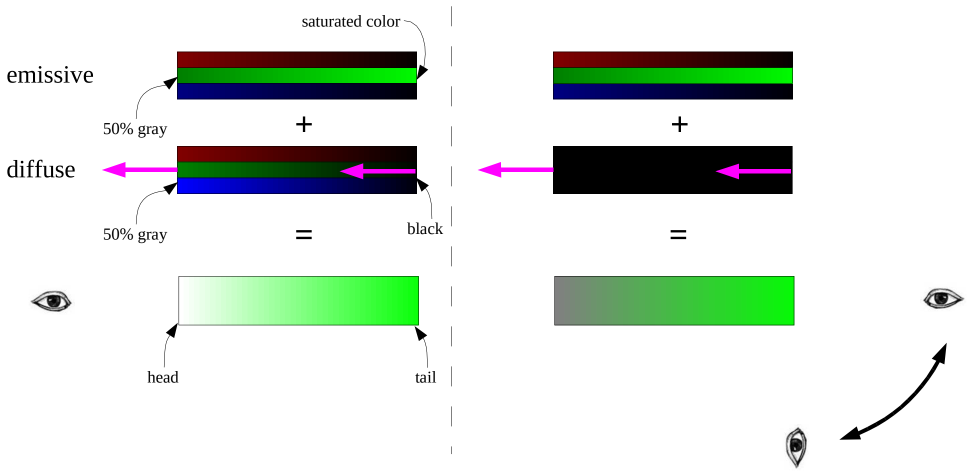


Figure 8: Comets lit by OpenGL Lighting.

If we let $L_e = 0.5$ and $k_d L_d = 0.5$, (2) resembles (1), with the exception of the max function. This suggests that if we dedicate a light source to represent the viewing direction (a "headlight"), we could use OpenGL lighting to illuminate comets even when they were statically placed in a display list.

Figure 8 shows how this works. The left diagram shows the case for vectors (i.e. normals) directed towards the viewer. The right diagram shows the case for vectors directed away from the viewer. Note that the left case obeys (1), while the right case shows the effect of the max function in (2). We have tried a number of approaches to overcome this, but to no avail. Nevertheless, we feel that the fact that comets in display lists work for oncoming directions still has some value. Applications that want to show diverging vectors accurately can abandon display lists and redraw the comets with comet lighting disabled when camera motion is complete, for instance: when trackball motion is ended.

4.3 Optimization

Our initial implementation of comets displayed some unusual behavior in the presence of graphics acceleration hardware. Some graphics cards gave rise to very poor comet performance relative to others.

Upon investigation, we found that the bottleneck lay in the fact that since comet appearance is view-dependent, our code was inquiring the modelview matrix every time a vector was drawn. In addition, it was pushing and popping line attributes in order to preserve any user-defined state outside of the library. Interestingly, the slowdown from each of these activities was significant.

think of a use for this!

Platform	optimized	unoptimized
PC	2,420	560
SGI	1,316	23
SUN	760	206

Table 4: Performance Comparison of N-Body Simulation With and Without Optimizations (in frames per second). For platforms' keys see Table 1.

To work around this difficulty, we enhanced the GLUV specification to include the functions *gluvBegin()* and *gluvEnd()*, which would act like the OpenGL functions *glBegin()* and *glEnd()*: The user calls *gluvBegin()* before drawing any vectors and *gluvEnd()* after the last vector is drawn, with the understanding that no modifications to the modelview matrix or any line attributes can be made in between.

Table 4 shows how this optimization improved performance. The differences in all cases are significant.

5 Results

Vector glyphs are not always displayed by themselves. In particular, they are often shown with "body glyphs" which encode positions of the relevant objects being simulated: molecules, charged particles,

	Hardware characteristics				
Platform	CPU	Speed	RAM	GPU	OS
PC	AMD Duron	1.2 GHZ	512 MB	nVidia GeForce3	Linux 2.4
SGI	8 R12000	400 MHz	8 GB	InfiniteReality3	Irix 6.5
SUN	2 SPARC Ultra 2	200 MHz	356 MB	SunVideo 1.4	SunOS 5.8

Table 1: Platforms tested.

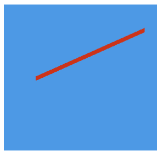
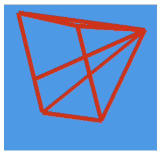
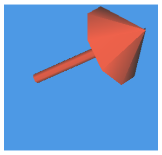
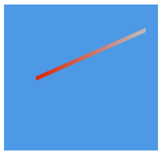
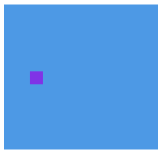
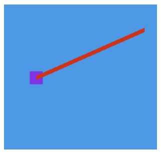
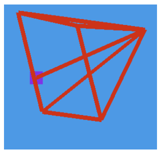
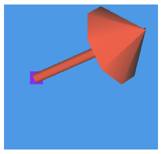
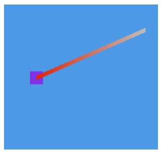
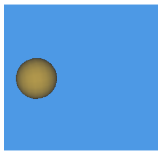
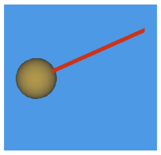
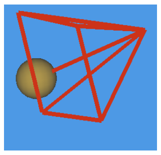
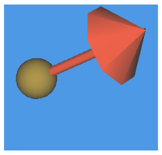
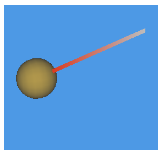
	Vector Glyph				
Body Glyph	None	Line	Pyramid	Cone	Comet
None	N/A				
Point					
Sphere					

Table 2: Vector and Body Glyphs.

stars, galaxies, etc. We have chosen sets of what we believe are popular vector and body glyphs, and include comets in the former. These are shown in Table 2.

In this section, we will study these glyphs systematically. We classify our results into qualitative and quantitative categories.

5.1 Qualitative

Qualitative evaluation criteria are generally subjective. However, it should be self-evident that the use of comets increases the simplicity in the visualization by reducing the amount of geometry on the screen compared with other all other glyphs except lines. When working with huge data sets, using comets significantly decreases the visual clutter and thus increases the visibility of important features needed for analysis.

The intuitiveness of comets is enhanced both by increased numbers of them in a single image and by the presence of body glyphs. Contrast the explanatory examples of a single vector glyph in Figure 5 with those of Figures 6 and 7. Although we cannot show it here, it is also enhanced in animation.

5.2 Quantitative

In order to assess the performance overhead involved in rendering vectors with comets, we wrote a small n-body gravitational simulation program which displays its rendering speed in frames per second and enables its user to change both vector and body glyphs to be all of the types shown in Table 2.

Table 3 shows the performance results on various platforms. There is a cell in the table that represents nothing (body and vector glyphs are both “None”) being displayed on the screen. This represents the overhead of the n-body calculations themselves.

Not surprisingly, on all platforms line glyphs were rendered fastest. Comets were next in decreasing mean performance, with speed reductions compared to lines of between 0% and 62%, except for point body glyphs on the SGI, where pyramids were drawn at a 17% speed reduction while comets had a 45% speed reduction. On all other platforms, comets outperformed pyramids and cones.

Increasingly-complex body glyphs make performance differences for varying vector glyphs negligible. Presumably, this same conclusion would apply to other non-vector-related graphics output.

6 Conclusions and Future Work

Comets are a useful vector glyph to have in the visualization developer’s toolbox. Our final results met our initial criteria for them. They reduce the crowding problem, eliminate the projection ambiguity, and provide high graphics performance. Additionally, we have shown how to put comets in display lists when such lists are desirable. Comets are intuitive: They do not require a steep learning curve for the viewer to extract visual information from them.

The one shortcoming of putting comets in display lists stems from the nature of the OpenGL lighting model. With the advent of programmable shading hardware on most platforms, we intend in future to write a slightly modified OpenGL shader to permit negative direction cosines and thus address this problem.

n-body performance						
	Vector Glyph					
Body Glyph	None	Line	Pyramid	Cone	Comet	**Platform**
None	14,500	4,300	806	273	2,420	PC
	1,659	1,445	1,194	516	1,316	SGI
	11,500	1,990	148	20	760	SUN
Point	6,500	3,000	715	265	2,100	PC
	1,615	1,416	1,180	515	780	SGI
	1,950	1,040	136	20	540	SUN
Sphere	40	40	38	35	40	PC
	128	127	120	104	117	SGI
	11	10	10	7	10	SUN

Table 3: Performance of an N-Body Simulation on Different Platforms (in frames per second). For platforms' keys see Table 1.

Availability

Contact any of the authors for source code and examples.

Acknowledgments

This research was performed in part using the Molecular Science Computing Facility (MSCF) in the William R. Wiley Environmental Molecular Sciences Laboratory, a national scientific user facility sponsored by the U.S. Department of Energy's Office of Biological and Environmental Research and located at the Pacific Northwest National Laboratory. Pacific Northwest is operated for the Department of Energy by Battelle.

References

BANKS, D. C. 1994. Illumination in diverse codimensions. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, ACM, 327–334.

BORING, E., AND PANG, A. 1996. Directional flow visualization of vector fields. In *Proceedings of the conference on Visualization '96*, IEEE Computer Society Press, IEEE Computer Society Press, 389–392.

DOLEISCH, H., AND HAUSER, H. 2002. Smooth brushing for focus+context visualization of simulation data in 3d. In *The 10-th International Conference in Central Europe on Computer Graphics*, Research Center for Virtual Reality and Visualization, 147–154.

DOVEY, D. 1995. Vector plots for irregular grids. In *6th IEEE Visualization Conference*, IEEE, 248–253.

FOLEY, J. D., VANDAM, A., FEINER, S. K., AND HUGHES, J. F. 1990. *Computer Graphics: Principles and Practice*, 2 ed. Addison Wesley, 75 Arlington Street, Suite 300, Boston, MA, USA.

HAUSER, H., LARAMEE, R. S., AND DOLEISCH, H. 2002. State-of-the-Art Report 2002 in Flow Visualization. Tech. rep., VRVis Research Center, www.VRVis.at, Feb. TR-VRVis-2002-003.

JOHANNSEN, A., AND MOOREHEAD, R. 1994. Visualization of mesoscale flow features in ocean basins. In *Proceedings of the Conference on Visualization*, IEEE Computer Society Press, Los Alamitos, CA, USA, R. D. Bergeron and A. E. Kaufman, Eds., IEEE Computer Society Press, 355–358.

KILGARD, M. J. 1996. *OpenGL Programming for the X Window System*, 1 ed. Addison-Wesley Pub Co, 75 Arlington Street, Suite 300, Boston, MA, USA.

KLASSEN, R. V., AND HARRINGTON, S. J. 1991. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proceedings of the Conference on Visualization*, IEEE Computer Society Press, Los Alamitos, CA, USA, IEEE Computer Society Press, 148–153.

KNIGHT, C. 2001. Visualisation effectiveness. In *2001 International Conference on Imaging Science, Systems, and Technology (CISST 2001)*, CSREA, 249–254.

LODHA, S. K., PANG, A., SHEEHAN, R. E., AND WITTENBRINK, C. M. 1996. UFLOW: Visualizing uncertainty in fluid flow. In *IEEE Visualization '96,*, R. Yagel and G. M. Nielson, Eds., IEEE, 249–254.

MAX, N., CRAWFIS, R., AND GRANT, C. 1994. Visualizing 3D velocity fields near contour surfaces. In *IEEE Visualization '94 Proceedings*, IEEE Computer Society, IEEE, 248–255.

NEIDER, J., DAVIS, T., AND MASON, W., Eds. 1993. *OpenGL Programming Guide*, 1 ed. Addison Wesley, 75 Arlington Street, Suite 300, Boston, MA, USA.

PANG, A., AND WITTENBRINK, C. 1997. Collaborative 3D visualization with CSpray. *IEEE Computer Graphics and Applications 17*, 2 (Mar.-Apr.), 32–41. ISSN 0272-1716.

PANG, A. T., WITTENBRINK, C. M., AND LODHA, S. K. 1997. Approaches to uncertainty visualization. *The Visual Computer 13*, 8, 370–390. ISSN 0178-2789.

REZK-SALAMA, C., HASTREITER, P., TEITZEL, C., AND ERTL, T. 1999. Interactive exploration of volume line integral convolution based on 3d-texture mapping. In *Proceedings of the conference on Visualization '99*, IEEE Computer Society Press, IEEE Computer Society Press, 233–240.

SCHROEDER, W. J., AVILA, L. S., AVILA, R., AND LAW, C. C. 1996. The design and implementation of an object-oriented toolkit for 3D graphics and visualization. In *IEEE Visualization '96*, R. Yagel and G. M. Nielson, Eds., IEEE, 93–100.

SCHROEDER, W. J., MARTIN, K. M., AND LORENSEN, W. E. 1998. *The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*, 2 ed. Prentice Hall PTR, Upper Saddle River, NJ 07458, USA.

SPIRKOVSKA, L., AND LODHA, S. K. 2002. Systems: AWE: aviation weather data visualization environment. *Computers and Graphics 26*, 1 (Feb.), 169–191.

WEINSTEIN, D. M., KINDLMANN, G. L., AND LUNDBERG, E. C. 1999. Tensorlines: Advection-diffusion based propagation through diffusion tensor fields. In *IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., IEEE, 249–254.

WEST, J., AND MACHIRAJU, R. 1998. A numerical imaging approach to comparative visualization. Tech. rep., Corps of Engineers, Waterways Experiment Station, Vicksburg, MS. Technical Research Report 98-18.

WITTENBRINK, C., SAXON, E., FURMAN, J., PANG, A., AND LODHA, S. 1995. Glyphs for visualizing uncertainty in environmental vector fields. In *SPIE & IS&T Conference Proceedings on Electronic Imaging: Visual Data Exploration and Analysis*, SPIE.

WITTENBRINK, C., PANG, A., AND LODHA, S. 1996. Glyphs for visualizing uncertainty in vector fields. *IEEE TRansactions on Visualization and Computer Graphics 2*, 3 (Sep), 266–279.