3D Vector Field Distribution Visualization with Glyphs

Xin Tong* The Ohio State University
Chris Jacobsen † The Ohio State University
Huijie Zhang ‡ Northeast Normal University, China
Han-Wei Shen § The Ohio State University
Patrick McCormick ¶ Los Alamos National Laboratory

ABSTRACT

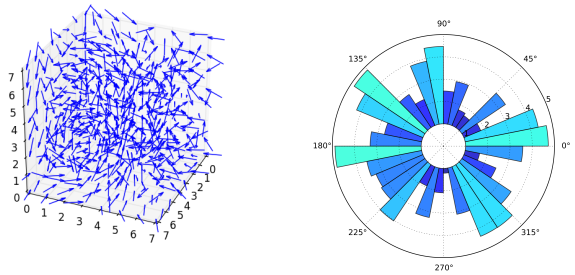
The major challenges of visualizing large-scale three-dimensional (3D) vector fields are the impact of visual clutter and an intensive computational workload. Visualizing vector fields with geometry (streamlines) or textures (LIC) requires the computation of integrals over a large memory footprint and often involves random, cache unfriendly, data access patterns. Large-scale data often contains noise and/or uncertainty that affects the quality of the visualization by producing visual clutter that interferes with the interpretation and identification of important features. A more ideal approach is to present statistics or distributions that help to describe the probability density of vector orientations. This paper presents the cube map histogram a new data structure for storing the distribution of 3D vectors, how to visualize the histogram with superquadric glyphs, and a hierarchical structure of the partitioned space to assist in the placement of the glyphs.With this representation users can identify the distribution and uncertainty of the vector field in an arbitrary axis-aligned rectangular region from the shape and color on the surface of the glyph. Users can also leverage the binary tree representing the partitioning to select the glyphs at different levels-of-detail (LOD) and achieve focus and context during the exploration of the data.

Keywords: Flow visualization, distribution, glyph

1 INTRODUCTION

The major three-dimensional flow visualization techniques include texture-based and geometry-based visualization. Line integral convolution (LIC) is a common texture-based visualization that displays a texture whose patterns follow the flow directions. LIC only provides flow patterns on a defined two-dimensional surface and thus it looses critical information about the flow in other spaces. Streamlines are a common geometry-based visualization that shows the trajectories of particles generated by integrating the vector directions across the 3D domain. This requires the tracing of particles within the flow and large datasets typically suffer a performance impact due to random data accesses across the memory hierarchy (including storage systems). Furthermore, even if streamlines can be generated for a large dataset, they often produce a cluttered display making it difficult to reason about important features and characteristics of the flow.

Glyph-based visualization allows users to quickly perceive the pattern of the multivariate data in the context of a spatial relationship [2]. A 3D vector is ideal to be visualized as a glyph because its three x, y and z components can be treated as multivariate data and are also related to its location in the spatial domain. This technique is leveraged in the conventional use of arrow plots for flow visualization. This technique places arrow glyphs throughout the space to show the vector field directions. While this approach gives users a direct understanding of the flow direction at a particular point, it suffers from congestion and occlusion as shown in Figure 1a. Displaying fewer glyphs can reduce such clutter but additional steps must be considered to avoid removing crucial information from the dataset.



(a) Arrow plots (b) Polar histogram

Figure 1: (a) The flow visualization with arrow plots suffers from visual clutter. (b) A polar histogram visualizes the distribution of 2D vector directions.

An additional way to incorporate more information into a glyph-based visualization is to present more information than just vector direction. For instance, a glyph can show the distribution of the flow directions in a local neighborhood. In a 2D vector field, this glyph can be a one-dimensional polar histogram [1] as shown in Figure 1b. A polar histogram segments the directions of the two-dimensional vector field into bins of equal one-dimensional angles, and it uses the 2D vector's possibility of falling into a bin as the height or the color of the bin. From the polar histogram, users know the distribution of the flow directions. Note that the polar histogram does not record the magnitude of the vector field. However, designing a histogram that represents the three-dimensional vector field distribution remains unsolved.

In this paper, we first introduce the cube map histogram as a model to discretize 3D vector directions and a data structure to efficiently and accurately store the 3D vector field distribution. We then visualize the cube map histogram using a three-dimensional glyph, which shows the distribution of vector directions in a local neighborhood. We design the glyph as a superquadric geometry, whose shape and orientation follow the principal component analysis results of the local vector field. In addition, an environment mapping of the cube map histogram is applied as a texture to the glyph's surface. Furthermore, the fatness of the glyph represents the uncertainty of the flow field, while the color distribution on the glyph tells the exact vector field distribution. Finally, the size of the glyph is scaled to fit its represented cubic block so that users know the space it corresponds to within the flow field. In order for the

*e-mail: tong@cse.ohio-state.edu
†e-mail: jacobsen.44@buckeyemail.osu.edu
‡e-mail: zhanghj167@nenu.edu.cn
§e-mail: hwshen@cse.ohio-state.edu
¶e-mail: pat@lanl.gov

glyph-based visualization to effectively convey maximal information about the data, we provide a top-down volume partitioning algorithm to divide the volume into a hierarchical tree structure based on the cube map distribution such that each glyph represents a spatial block. To provide a user-friendly interactive data exploration system, we allow users to interactively explore the treemap to focus on specific glyphs (regions) without occluding or querying the tree structure in order to inspect the partition results at different levels of detail (LOD).

2 Related Works

The traditional use of vector field histograms has been applied to real-time obstacle avoidance in mobile robots [1]. This technique only uses a two-dimensional vector field as input and our work extends it to support three-dimensional vectors. Leopardi [16] proposes an algorithm to partition the surface of the unit sphere into equal area regions that can be used to represent the histogram bins of the 3D vector field. Our proposed method also partitions the sphere's surface but uses less computation and leverages the cube map texturing capabilities in modern GPU architectures to achieve higher rendering performance.

Glyphs are very powerful in two-dimensional visualization of velocity in flow fields when there are no concerns of visual clutter or occlusion. G. Kirby and Laidlaw [14] provide a 2D flow visualization that uses arrow glyphs to represent velocity and ellipse glyphs to show the rates of strain tensors. They blend multiple layers of semi-transparent glyphs and background colors together to simultaneously display multiple data attributes. Peng and Larameee [19] use two-dimensional glyphs to visualize the flow on the surfaces of an unstructured adaptive resolution boundary mesh. With a fast image-based glyph placement algorithm, users can interactively change the glyph resolution to support a multi-resolution visualization. In 3D flow visualization, the vector glyph is used as a simple and direct rendering of the local vector field [5, 6]. The impact of visual clutter and the occlusion of important details become important factors in determining the effectiveness of these glyph-based techniques. The 2D projections of 3D glyphs can overlap if they are not well placed in the 3D space. Boring and Pang [3] apply different lighting conditions to a three-dimensional hedgehog glyph and other geometry-based forms of the flow to emphasize different vector directions. By only highlighting the glyphs corresponding to user-defined directions, they reduce the displayed data and hence alleviate the clutter problem. Laramee [15] addresses visual clutter by resampling the vector field and generating a smaller number of summary vectors that leads to a sparser glyph placement. In addition to the flow velocity, glyphs have also been used in visualizing other features, such as vortices. Sadarjoen *et al.* [21] detects vortices and fits each vortex with an ellipse or ellipsoid icon to illustrate both its shape and orientation. Reinders *et al.* highlight that elliptical shapes are not always a good description of the shape of a vortex structure [20]. Instead, they extract a skeleton graph from the vortex structure and visualizes it using cylindrical icons.

Our work is closely related to vector field uncertainty glyphs that use techniques to show the uncertainty information of local regions within a vector field. Lodha *et al.* [17] place uncertainty glyphs along particle traces to show the magnitude of the deviation between two streamlines. Wittenbrink *et al.* [23] presents a technique that expresses the variation of both vector direction and magnitude using different glyph shapes. Zuk *et al.* [24] discusses the interactive rendering of glyphs for visualizing uncertainty in a bidirectional vector field. Hlawatsch *et al.*[10] extends the vector field uncertainty glyph to address unsteady flows. This technique shows the possible ranges of flow directions by angles and the time by changing the glyph radius. All of the above techniques demonstrate their effectiveness when placed on either a slice plane or on a surface but do not address the visual challenges of representing the entire 3D volume. More specifically, these techniques all use a flat two-dimensional shape even when displayed within a 3D space – thus loosing the details of the three-dimensional flow ambiguous. Additionally, vector direction variation or range information presented using these techniques is only a summary of uncertainty. The glyphs used in our approach provides the distribution of 3D vector directions that also contains more details about the uncertainty and other important information for fully interpreting details of the flow.

Our design is inspired by tensor glyphs, which play an important role in tensor field visualization. Westin *et al.* [22] derives the geometric anisotropy measure from the tensor field values and then uses them to generate different shapes of ellipsoids to visualize the field. Kindlmann [12] improves the ellipse glyph design with a superquadric shape. This superquadric overcomes the problems of asymmetry and visual ambiguity in the previous designs. Kindlmann and Westin [13] optimized the tensor glyph placement technique by densely packing the glyphs using a particle system. In this way, the glyph distribution does not have the pattern of the original data sampling grid and appears more continuous.

3 3D Vector Field Distribution

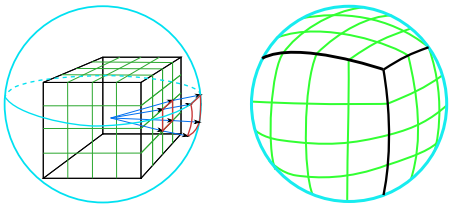
A histogram for 3D vector direction records the distribution of a set of 3D vector directions. A bin in this histogram represents a range of similar 3D vector directions. The size of the bin is a solid angle, a two-dimensional angle in three-dimensional space. Given a fixed number of bins, the bins of the 2D vector histogram have the same fan shape as shown in Figure 1b. But for 3D vector directions, the bins can have different shapes depending on the way that the solid angles are partitioned, as mentioned in the related works. In this work, we design our histogram using the concept of the cube map in computer graphics environment mapping. The cube map histogram ensures that the solid angle covered by each bin has a relatively equal size, while achieving fast determination of bins for a given 3D Cartesian coordinates vector as described in Section 3.1. After normalizing the histogram bins with their solid angles as talked in Section 3.2, the histogram bin value reflects the true frequencies of 3D directions.

3.1 Cube Map Histogram

The problem of binning the 3D vector field directions into equal solid angle ranges is equivalent to dividing the sphere surface into small equal area patches because we know the solid angle in radian is equal to the area of its projected patch on a unit sphere. There are existing methods to partitoin the sphere into equal area pathces, such as using spherical polyhedron and Leopardi [16]'s method. However, the spherical patches we need should also be easy to become histogram bins, which means a vector starting from the sphere center should easily know which patch it intersects. If using spherical polyhedron, we need to both store the polygon mesh and perform intersection between a ray and a polygon, which are not cheap. If using Leopardi's method, we need to convert our vector in Cartesian coordinates to spherical coordinates using trigonometric functions, which is not cheap in computer either.

Our binning can be created by projecting a cubic uniform grid onto a sphere surface. Assume there is an inscribed cube of the unit sphere, who has each face divided into regular grids as shown in Figure 2a. If projecting the grid on the cube from the cube center towards the sphere surface, we get a grid on the surface as well, as shown in Figure 2b. The points on two grids then have one-to-one correspondence. We use the grid cell as the histogram bin for 3D vector directions. From another perspective, if placing a 3D vector at the sphere center, the vector belongs to the bin of the cell that it intersects with.

This theory of projecting a 3D vector onto a cube map and determine projected location on the cube map is computationally sim-



(a) (b)

Figure 2: (a) The projection between a patch on the sphere surface and a patch on the surface of the inscribed cube. (b) Cube map grid on the sphere surface.

Table 1: Binning

Condition	b_f	b_x	b_y
$v_z < -\lvert v_x\rvert$ & $v_z \le -\lvert v_y\rvert$	0	$\lfloor(1-\frac{v_x}{\lvert v_z\rvert})\cdot\frac{d}{2}\rfloor$	$\lfloor(1-\frac{v_y}{\lvert v_z\rvert})\cdot\frac{d}{2}\rfloor$
$v_x < -\lvert v_y\rvert$ & $v_x \le -\lvert v_z\rvert$	1	$\lfloor(1-\frac{v_y}{\lvert v_x\rvert})\cdot\frac{d}{2}\rfloor$	$\lfloor(1+\frac{v_z}{\lvert v_x\rvert})\cdot\frac{d}{2}\rfloor$
$v_y \le -\lvert v_z\rvert$ & $v_y < -\lvert v_x\rvert$	2	$\lfloor(1+\frac{v_z}{\lvert v_y\rvert})\cdot\frac{d}{2}\rfloor$	$\lfloor(1+\frac{v_x}{\lvert v_y\rvert})\cdot\frac{d}{2}\rfloor$
$v_z \ge \lvert v_x\rvert$ & $v_z > \lvert v_y\rvert$	3	$\lfloor(1+\frac{v_x}{\lvert v_z\rvert})\cdot\frac{d}{2}\rfloor$	$\lfloor(1+\frac{v_y}{\lvert v_z\rvert})\cdot\frac{d}{2}\rfloor$
$v_x \ge \lvert v_y\rvert$ & $v_x > \lvert v_z\rvert$	4	$\lfloor(1+\frac{v_y}{\lvert v_x\rvert})\cdot\frac{d}{2}\rfloor$	$\lfloor(1-\frac{v_z}{\lvert v_x\rvert})\cdot\frac{d}{2}\rfloor$
$v_y > \lvert v_z\rvert$ & $v_y \ge \lvert v_x\rvert$	5	$\lfloor(1-\frac{v_z}{\lvert v_y\rvert})\cdot\frac{d}{2}\rfloor$	$\lfloor(1-\frac{v_x}{\lvert v_y\rvert})\cdot\frac{d}{2}\rfloor$

ple, and is covered in the original environment mapping paper [9]. However, we use a different coordinate systems in our technique which ensures better consistency on the boundary of the faces. Thus, we explain the cube map algorithm below again using our own coordinate system. For a given vector in Cartesian coordinates $v = (v_x, v_y, v_z)$, its bin is represented as (b_f, b_x, b_y), where b_f is an integer index of the cube face in the range of $[0,5]$, and b_x and b_y are the integer indices of the grid on one face in the x and y directions. Table 1 gives how to convert a 3D coordinate (v_x, v_y, v_z) to the 3D bin index (b_f, b_x, b_y), where d is the dimension of the 2D grid on each face. The dimensionality of the bin index can be further reduced to 2D as $(b_f \cdot d + b_x, b_y)$ when rendering the histogram as a 2D image, or be reduced to 1D as $b_f \cdot d^2 + b_x \cdot d + b_y$ when storing in the linear memory space.

From the equations in the table, we see that the computation cost of the bin index for a 3D vector is very small comparing to other two methods mentioned above. Minimizing this computation is very important for building histograms of large-scale datasets due to the large number of cells.

3.2 Histogram Normalization

Histogram describes a distribution only when its bin sizes are equal. Even though the corresponding solid angle of our histogram bins are close to equal, they are still not exactly equal. So it is not accurate to use bin counts computed by Section 3.1 to represent the distribution frequency. In order to correct it, we can normalize the bin counts by dividing it by the corresponding solid angle. In the following, we describe how to compute the solid angle of each bin.

Figure 3a shows a face of the inscribed cube, which is divided into 4×4 cells. Each cell represents a histogram bin. From the figure, we can see four rectangular patches marked as red, yellow, green and blue frames, which are all centered at face center O. We can imagine that connecting an arbitrary rectangular patch with the cube center forms a four-sided rectangular pyramid. Because the center of these four pyramid base are all at O, this pyramid is a right rectangular pyramid. For a right rectangular pyramid as in Figure 3b, its solid angle can be computed by:

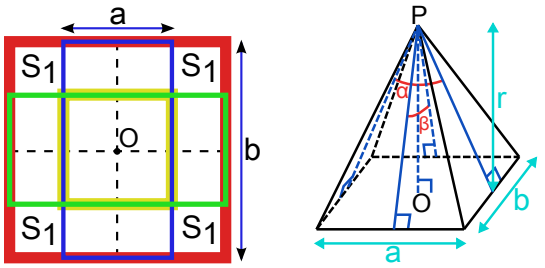
$$S = 4 * arcsin(sin(\frac{\alpha}{2}) * sin(\frac{\beta}{2})) \tag{1}$$

where α and β are the apex angles of the pyramid.

The apex angles of the pyramid can be computed from the edge lengths of the rectangle. For example, for the blue rectangle in Figure 3a, its two edge lengths are a and b. From Figure 3b, we can compute the two apex angles by:

$$\alpha = 2 \cdot \arctan(\frac{a}{2 \cdot r}) \tag{2}$$

$$\beta = 2 \cdot \arctan(\frac{b}{2 \cdot r}) \tag{3}$$



(a) (b)

Figure 3: (a) Four patches colored as red, green, blue, and yellow. (b) Right rectangular pyramid, whose base is a $a \times b$ rectangle and height is r, apex angles are α and β.

where r is the distance between the cube center and the cube face. Then Eq. 1 becomes

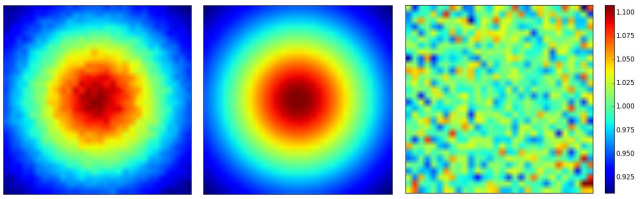
$$S = 4 \cdot arcsin(sin(\arctan(\frac{a}{2 \cdot r})) \cdot sin(\arctan(\frac{b}{2 \cdot r})) \tag{4}$$

Next, we need to compute the corresponding solid angle for each of the 4×4 cells in Figure 3a. For example, we pick the cell at the corner of the face as an arbitrary cell, marked as S_1 in Figure 3a, and compute its solid angle. The four cells are symmetric and have the same solid angle S_1. Because those four cells are not centered at the cube face center O, its connection with the cube center wouldn't form a right rectangular pyramid. But S_1 can be computed by using the solid angles of four other rectangles centered at O, shown as red, green, blue, yellow frames in Figure 3a as:

$$S_1 = \frac{S_{red} - S_{green} - S_{blue} + S_{yellow}}{4} \tag{5}$$

Then the four solid angles, S_{red}, S_{green}, S_{blue}, S_{yellow}, can be easily computed by Eq. 4. For an arbitrary bin (b_f, b_i, b_j), we can always find such a set of four rectangles to compute its solid angle. Because the six faces are symmetric, we only need to compute the solid angles for one face and reuse it on the other 5 faces.

Figure 4b uses color to visualizes the solid angles of the bins in one cube map face. We notice that the bins around the center have larger solid angles while the bins around the boundary have smaller solid angles. To verify the correctness of using the solid angle to normalize the cube map histogram, we randomly generated a large number of 3D vectors. If our generated distribution is close to a uniform distribution, our normalization can be proved as correct. Figure 4a gives one face of the cube map histogram for a uniform samples before normalization. After normalizing it by the solid angle shown in Figure 4b, we get the normalized histogram Figure 4c, which is mostly around the value of 1 with small variations.



(a) (b) (c)

Figure 4: (a) Histogram of uniform random sample. (b) solid angles for the cells on one cube face. (c) Normalized histogram by the computed solid angles.

The variation is from the error of the uniform sampling, which can be ignored. This confirms the correctness and necessity of the cube map histogram normalization.

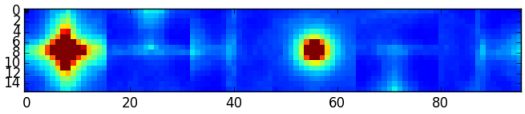


Figure 5: Images of the 6 faces of the cube map

We compute and display the cube map histogram of the Solar Plume dataset as a 2D image shown in Figure 5. Solar Plume is a simulation of the solar plume on the surface of the Sun with a resolution of $126 \times 126 \times 512$. From this figure we can tell that the $-z$ direction ($b_f = 0$) and the $+z$ direction ($b_f = 3$) on the histogram have high probability.

4 Vector Field Distribution Visualization

In the previous section, we plot the computed the vector field distribution on a 2D image in Figure 5. However, it is not intuitive to do so, because of lacking the correspondence between the bin and the represented 3D vector direction. Instead, we should visualize the histogram in the context of the 3D data space and let the histogram bins point to their represented 3D directions. In this section, we propose the design of our superquadric glyph that visualizes the histogram. Besides, we place our glyphs in the data volume based on a hierarchical space partitioning algorithm and place a glyph inside each block.

4.1 Glyph Design

A vector field histogram consists of many data variables because we can consider each bin as a data variable. Besides, each bin represents a vector direction in the space domain. A glyph generally has 6 major visual channels: color, shape, size, orientation, texture, and opacity [2]. Mapping a large number of data variables onto a limited number of visual channels is challenging.

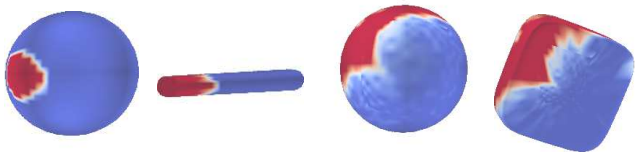
Table 2: Correspondence between visual channels and data attributes

Visual channel	Data attribute	Relationship
color & texture	histogram bin values	color map
shape	principal components	eigenvalue guided superquadric
size	represented block size	fit glyph tightly inside a block
orientation	principal components	eigenvector directions

We use a solid 3D shape, such as a sphere or a superquadric, with color texture on its surface as the design of our glyph, as shown in Figure 6. For each point on the glyph surface, its connection to the glyph center gives us a 3D vector direction. We relate this direction to the represented 3D direction of a bin in the 3D cube map histogram. In other words, there is a one-to-one correspondence between each bin in the cube map histogram and each patch on the glyph surface. We use different colors or shapes on this patch to express the value of the bin in the cube map histogram. We list the correspondence between the visual channels of our glyph and their represented data attributes in the Table 2. In the following, we will show how we encode the distribution onto the visual channels.

4.1.1 Color on Glyph Surface

Color is mapped on the glyph surface to express the the bin value by following a colormap. As shown in Figure 6, we use a sphere as the shape of the glyph and map the bin values of the histogram onto the sphere with the cool-warm colormap. Technically, histogram values are converted to RGB values using color map, and stored in the OpenGL cube map texture. Every vertex on the sphere surface is given a texture coordinate, which can be the corresponding histogram bin's represented 3D vector direction. Then the OpenGL will render the sphere with the cube map texture on the surface. If a certain patch on the sphere has a red color, it means the corresponding vector direction has high frequency on the histogram. The limitation of using color is occlusion. Half of the glyph surface is invisible unless rotating the glyph. Section 4.3 will give more details about the color map and the shading on the glyph.



(a) D_1 sphere (b) D_1 superquadric (c) D_2 sphere (d) D_2 superquadric

Figure 6: Use sphere and superquadric as the glyph shapes for two different distributions D_1 and D_2.

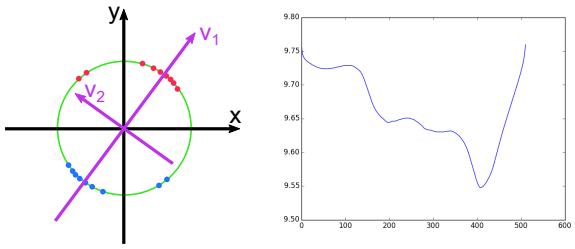
4.1.2 Superquadric Glyph Shape

Because of occlusion and visual clutter, it is not possible for users to perceive the entire distribution from the color texture from a fixed view point. Thus we use another two visual channels, shape and orientation, to convey more distribution information that is the overall trend of the distribution. One simple way of using the shape is to extrude the mesh of the previously mentioned spherical glyph from its center and use the bin value as the amount of the extrusion. However, this will give us an arbitrary shape that may have a very rough surface when the histogram is noisy and can also cause severe occlusion. Thus, we want to use a shape that is more descriptive than

the sphere and also relatively simple. Instead of encoding the entire histogram information on the glyph shape, we choose to only encode its principal vector directions from principal component analysis (PCA). Kindlmann[12] used superquadric tensor glyph to visualize the eigenvalues and eigenvectors of a tensor matrix. To utilize this technique, we can generate the covariance matrix from the original 3D vector field samples, and use the eigenvalues and eigenvectors of the matrix to build the superquadric shape as described in the Kindlmann's paper.

We use 2D vector field as example to describe how we use PCA. Figure 7a illustrates the process of PCA for a set of 2D vector directions. Red points represent the normalized 2D vectors and are placed on a unit circle. In order to keep the distribution center at the unit circle center, we add same number of blue sample points that are symmetric to the original red sample points without affecting the ei Then, eigenvalues and eigenvectors are generated from all sample points, including both red and blue points. In Figure 7a, v_1 is the principal vector direction or the eigenvector corresponding to the largest eigenvalue, which points to the largest population of the sample points. v_2 is the second principal vector direction and nearly points to the second largest sample population. Sometimes, people fit an ellipse to the two purple axes to describe the distribution, which becomes an ellipsoid in 3D case. Again, superquadric is a better alternative to ellipsoid as mentioned in the related works.

In 3D case, we can also perform PCA and will get three eigenvalues λ_1, λ_2, λ_3 ($\lambda_1 > \lambda_2 > \lambda_3$) that determine the shape of the superquadric, and three eigenvectors v_1, v_2, v_3 that determine the orientation of the superquadric. Figure 6 shows two different distributions (D_1 and D_2) using both sphere and superquadrics. The large values in the distribution D_1 distributed at a small size region as seen from the red regions in Figure 6a. So its largest eigenvalue λ_1 dominates, and is much larger than the other two, which makes the superquadric shape become more linear as shown in Figure 6b. On the other hand for distribution D_2, large values are distributed in a larger linear region as shown in Figure 6c. So its first two eigenvalues λ_1 and λ_2 are comparable, which makes the superquadric shape more planar as shown in Figure 6d.



(a) (b)

Figure 7: (a) Principal component analysis of the 2D vector directions of the sample points. (b) $H(N_{lr})$ values on different splitting points along the longest dimension.

4.2 Hierarchical Glyph Placement

The glyph can be placed in different locations of the volume with different sizes to represent the vector field distribution of different spatial regions. To express the flow distribution over the entire data space, we need to decide the location and size of the glyphs. Because the glyph is always a convex shape, we can partition the 3D volume into multiple rectangular blocks and fit a glyph to each block to represent the distribution inside. Our partition criteria is to make the vector field inside each block as coherent as possible. A top-down space partitioning approach works better than a bottom-up approach in terms of creating convex blocks. To partition our space, we recursively subdivides the volume until the vector field inside each block becomes coherent. The entropy of the cube map histogram to describe the coherence of the vector field, so that the subdivision stops when the entropy reaches a threshold.

More importantly, we need to find out where to place the cut to subdivide the volume. Because we only talk about a 3D regular grid dataset, we place a cut that is parallel to the one of the three axis-aligned planes so that the subregions after cutting are still a rectangular shape that can keep being cut. Additionally, because we want to get a round region that is close to the shape of our glyph, the cutting plane that is perpendicular longest block dimension is chosen. The number of options for placing the cut is the length of the longest dimension of this block. We want to be able to separate different flow directions to the two sides so that each side can be more coherent or pure. Here we use the concept of impurity from data mining theory to solve this optimal cut problem. In the binary decision tree algorithm, entropy can be used to describe impurity. Before the cut, the entropy of the block is $H(N)$. We cut the block into two smaller blocks with fractions of P_l and $(1-P_l)$, whose entropies are $H(N_l)$ and $H(N_r)$, respectively. After cutting, then the total entropy of the entire block is reduced. And the amount of the entropy reduction is defined by [7]:

$$\Delta H(N) = H(N) - P_l \cdot H(N_l) - (1-P_l) \cdot H(N_r) \qquad (6)$$

For a given block, the value of $H(N)$ is fixed. So in order to maximize the $\Delta H(N)$, we need to minimize the $H(N_{lr}) = P_l \cdot H(N_l) + (1-P_l) \cdot H(N_r)$, which is a weighted sum of the entropy of the cut's two sides.

A brute force method to find the minimum point is to compute $H(N_{lr})$ for every point along the longest dimension and find out the minimum value. For example, in the Solar Plume dataset, when placing the first cut, we compute and plot the $H(N_{lr})$ in Figure 7b. There is a very obvious minimum point near $x = 400$, which is the point to place the optimal cut.

An easy way to accelerate the computation of $H(N_{lr})$ is to reuse the previously computed histogram with a sliding plane perpendicular to the longest dimension. After computing histograms of the two blocks on the two sides, we shift the cutting plane, and then add or subtract the histogram from the plane on the histogram of the previously computed histograms to get the histogram of the new two blocks. An even faster implementation is to compute the histogram for each slice parallel to the cut plan, and then make a scan or prefix sum on the histograms of the slices to compute the histograms for different subregions at once. The prefix sum can be implemented with a parallel algorithm on a parallel machine to achieve tremendous speedup, which can be a future work.

4.3 Glyph Rendering

It is important to make the glyph's rendering coherent with the represented histogram values. As mentioned before, the values of the histogram are mapped onto the superquadric using a colormap. As we known, the rainbow colormap is very popular in scientific visualization. However, perceptual changes of the rainbow colormap are heterogeneous among the different colors, so that its nonuniform changes is unable to accurately reflect the uniform changes in our histogram values. In this paper, we adopted the cool-warm color map recommended in [18] as the default colormap. The cool-warm color map has two major color components, cool component and warm component. The transition between them is an intermediate white color. The colormap not only has a perceptually linear transition, but also is aesthetically pleasing and guarantees minimal interference with shading. In our case, we map the cool blue color to low histogram frequency values and warm reds to high values. To gain better depth cue for our 3D glyph rendering, the Blinn-Phong illumination model is adopted, so that users can clearly observe the

Table 3: Design guideline

Index	Guidelines	Our glyph matched
DG1	visualization space	Y
DG2	complexity vs. density	Y
DG3	hybrid visualizations	Y
DG4	perceptually uniform properties	Y
DG5	redundant mapping	Y
DG6	importance-based mapping	Y
DG7	view point independence	Y
DG8	simplicity and symmetry	Y
DG9	orthogonality and normalization	N
DG10	intuitive / semantical mapping	Y
DG11	balanced glyph placement	Y
DG12	facilitate 3D depth perception	Y
DG13	interactive occlusion control	Y

orientation of the superquadric shape and the spatial relationships among the glyphs.

4.4 Design Analysis

Borgo *et al.* [2] provides thirteen general considerations and guidelines for glyph design, as shown in Table 3, by summarizing a few previous glyph techniques. We followed these design guidelines to design our glyph and matched most of them. Note that it is not always possible or required to match all of them.

Now we describe how the design matches the guidelines one by one. **[DG1]**Because our visualization space is a 3D volume, our glyph is a 3D shape as well. **[DG2]** Our glyph is relatively complex because of the color texture, so we keep the glyphs density relatively low to make sure users are able to easily view the texture patterns on the glyph. **[DG3]** An optional rendering of the streamlines in the same space with the glyphs can provide spatial context and other additional information. **[DG4]** Our well designed cool-warm color map ensures that equal distance in the bin values is perceived equal in color difference, in contrast to the rainbow color map. **[DG5]** We map data variables (histogram bins) not only on the visual channels of the color texture, but also redundantly map the principal directions of the vector field on the visual channel of shape to reduce the risk of information loss and occlusion. **[DG6]** The bins with high values are more important and should be highlighted, so we make the sharper ends of the superquadric shape coincide with the high value bins. **[DG7]** The superquadric shape is symmetric with respect to three orthogonal axes, so its shape can be perceived with arbitrary view directions. **[DG8]** Our superquadric shape is simple, smooth, and symmetric. The symmetry makes it easy for users to mentally reconstruct the shape of the occluded parts of the glyph. **[DG9]** The visual channels can be perceived mostly independently, except a few points. First, the shape of the glyph (fat or thin) can slightly affect the perception of its size. Also, a bin that is distributed far from the glyph center may appear larger than the ones closer to the glyph center because of the distortion on the superquadric shape. **[DG10]** The shape and color mapping on the glyphs is intuitive because the sharper red ends of the glyph always point toward the major flow directions; the fatness of the glyph infers the uncertainty. **[DG11]** There is no obvious artificial pattern in the glyph placement based on our space partitioning algorithm, in contrast to placing glyphs on the regular grid points. Being able to display the glyphs of the blocks at different levels of the hierarchical partition alleviate the overlapping problem. **[DG12]** Shading effects on the glyphs can help to enhance the depth perception. **[DG13]** Allowing users to explore the data with a treemap visualization helps offset occlusion problems.

5 Spatial Exploration of Vector Distribution

Simply visualizing all the glyphs in the 3D spatial domain statically can result in visual clutter, especially when dealing with a large dataset with a large number of glyphs. The user needs some sort of way to find items of interest and see the entire set all at once in two dimensions. Thus we provide a treemap to visualize the nodes of the space partitioning tree, as well as their entropies and corresponding spatial volumes. Furthermore, we allow the user to explore the hierarchical structure of the tree by querying both globally and locally. The accompanying video demonstrates our interactive visualization system.

5.1 Glyph Query

To globally control the number of glyphs to render, our system allows users to control the granularity of the space partitioning by specifying the entropy threshold. Using different entropy threshold values in our space partitioning algorithm will result in different trees. A tree using a higher threshold is a sub-tree of a tree generated using a lower threshold; this is accomplished by trimming some of the nodes whose entropies are lower than the higher threshold. Thus if we have a bigger tree generated using a low enough threshold, we can easily generate arbitrarily smaller trees corresponding to any higher entropy threshold. In the glyph visualization, by specifying different entropy thresholds, we can immediately get the tree and render the glyphs corresponding to its leaf nodes. A higher threshold value results in less but larger superquadrics, while a lower value results in more but smaller superquadrics. In the system, when the user requests the entropy threshold increase or decrease, we iteratively increase or decrease this amount until a change is detected in the tree. This prevents the user from having to be aware of the entropy threshold values. We also allow the user to set the values more exactly through a slider if needed.

To allow local control of the glyphs, we allow users to click on a glyph to subdivide the corresponding block into two subregions and visualize the two glyphs corresponding to them. This is achieved by checking the tree node of the clicked glyph to see whether it has child nodes. If so, the two glyphs corresponding to the two child nodes are displayed in place of the original glyph. In this way, we allow the users to interactively split any glyph, expanding individual branches of the tree to show more nodes from the original tree. The users can thus drill into the data locally to explore more details in the region of interest.

5.2 Treemap Visualization

A treemap is a visualization introduced by Shneiderman and Johnson [11] that displays one rectangle for each of the leaf nodes in a tree. The area of each rectangle is proportional to its data value. This is often useful for finding data values that are larger. In our treemap, we show rectangles at different sizes based on the volume occupied by each superquadric's spatial block. It allows the user to quickly find the larger size blocks, which are more representative in the data volume.

Multiple layout algorithms can be used to build a treemap. The original layout algorithm used in Schneiderman and Johnson's paper can result in thin rectangles in the layout, which is not aesthetically pleasing and difficult to be picked by users. We instead used the squarify layout introduced by Bruls et al [4]. It provides good aspect ratios for our rectangles, which makes it much easier for users to identify the areas.

As shown in Figure 8, we also give colors to the rectangles based on the entropy values of the represented spatial blocks so that the users can use color to pinpoint the glyphs with larger or smaller entropy values. The ranges of colors are shown with a colorbar to the right of the widget. Selecting a rectangle in the treemap highlights the corresponding superquadric in the scene and vice versa,

allowing the users to correlate the spatial blocks between the two visualization spaces.

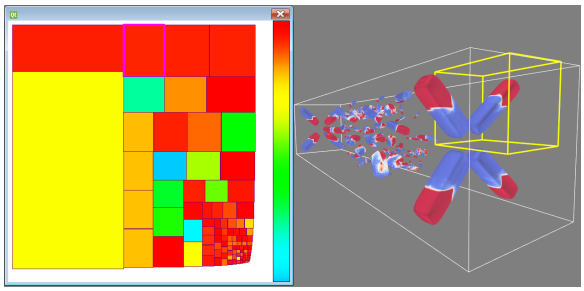


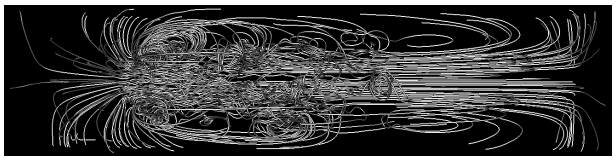
Figure 8: Treemap Visualization.

6 Case Studies

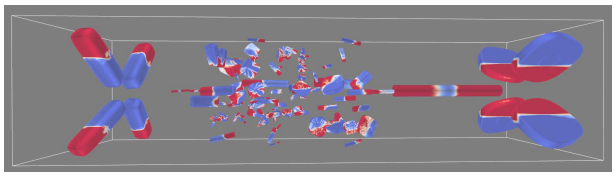
To explore the effectiveness of our technique we applied our algorithm to two different, uniform grid-based datasets. In this section we review each of these two datasets as well as provide a summary of our findings.

6.1 Solar Plume

The *Solar Plume* dataset is generated from a simulation of the plume on the surface of a sun-like star. It has a resolution of $126 \times 126 \times 512$ and Figure 9 shows the data using both streamlines and our glyph-based approach. The beginnings of visual clutter and occlusion are apparent in the streamline-based image. As previously discussed, this makes it difficult to understand the overall flow distribution as it only shows the exact 3D direction at individual points.



(a) Streamline visualization for solar plume.

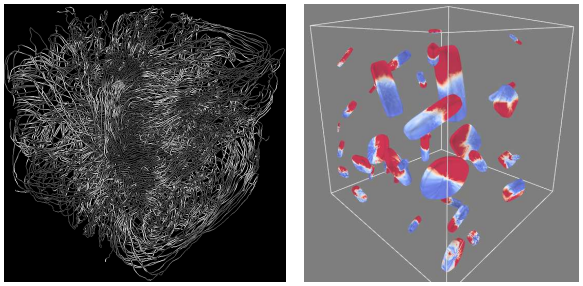


(b) Glyph visualization for solar plume.

Figure 9

In Figure 9b and the accompanying video, the superquadric glyphs of different sizes, shapes and orientations are displayed throughout the volume. Each glyph represents the vector field distribution of an axis-aligned rectangular space around it. The left-most four glyphs show a diverging flow structure as the red portion of the glyphs are all pointing towards the outside of the volume. In contrast, the right-most four glyphs show converging flow as the red part of the glyphs all point inward within the volume. In addition, the left four glyphs have linear shapes that represent a high certainty of flow direction. The right four glyphs each have a planar form, representing that it has another vector component other than its major vector direction. Specifically, the converging flows on the right also moves towards the left – towards the middle of the volume. In the middle-left region a large number of small glyphs indicate a turbulent region due to high entropy calculation and our space partitioning algorithm. In this region the glyphs in the center mostly point towards the left, while in the vicinity there are a few planar shape glyphs that represent potential vortices. Lastly, there is only one linear glyph on the middle-right region that has red on both ends. This captures that the vector distribution has two opposite components. Ideally, we would want to keep subdividing this region to separate the two flows. However, the current entropy calculation used in space partitioning cannot recognize the bin order within the histogram.

6.2 Nek



(a) Streamline for Nek. (b) Glyphs for Nek.

Figure 10

The *Nek* dataset is generated from the Nek5000 simulation code. As explained in the paper by Paul Fischer et al. [8], Nek5000 simulates thermal hydraulics in a nuclear reactor using the spectral element method. This allows the simulation of coolant flowing between and around the pins in the reactor. The dataset resolution is $128 \times 128 \times 128$.

Figure 10a shows the rendering of the dataset once again using streamlines as a comparison. The impact of visual clutter is significant in this dataset as the occlusion makes it impossible to visualize the full details of the flow. In contrast, the glyph-based version in Figure 10 shows the flow distribution much more clearly. In this case, we see a decomposition into glyphs throughout the entirety of the volume. This represents the overall amount of turbulence within the data that is also suggested by the occlusion issues within the streamline representation. The glyphs in the central region of the volume are big, indicating a more coherent flow, while the glyphs on the left are small and are thus relatively more turbulent. The large linear-shaped glyphs in the center are pointing upwards and are relatively fat, so we know the majority flow in the center goes up with a high variation (once again suggesting a turbulent flow). The glyphs close to the near-right plane are mostly planar and thus that the flow directions mostly reside on the corresponding plane. Additionally, from the texture applied to the bottom right glyph, we know the flow has components going left and going down along its corresponding plane.

7 Performance

We measured the performance of the computation and interactive visualization system on a machine running Windows 8.1 with an Intel Core i7-4700HQ CPU with 32 GB RAM and an nVidia GeForce 765M GPU with 2GB of frame buffer memory. We used the Nek dataset at different resolutions to measure the performance and scalability of our technique. The shape of our tested volume datasets are all cubic.

The blue bars in Figure 11 show the execution time of the cube map histogram computation using different numbers of vectors.

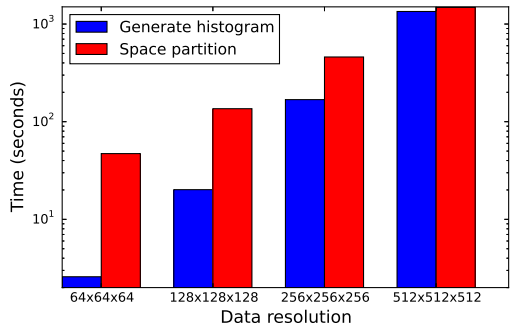


Figure 11: Time of generating cube map histogram.

This time includes computing the histogram bin indices of the vectors on all the cells using Table 1, normalizing the histogram with solid angles computed from Equation 5 and counting the bin indices. The last two operations take less than 1% of the total execution time. From the figure, we notice that the time increases almost linearly with an increasing number of input vectors.

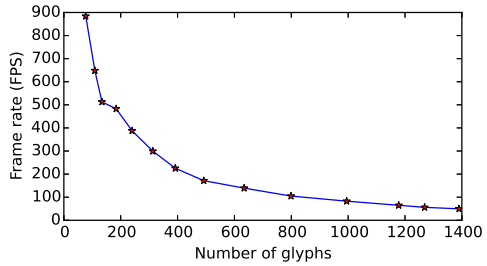


Figure 12: Frame rate.

The red bars in Figure 11 show the time required to compute the hierarchical space partitioning in support of visualizing the histogram data. This time includes recursively building the histogram from the computed bin indices and computing the entropies on the two sides of a splitting plane using the mentioned sliding plane method. In addition, it includes determining the splitting plane position with Equation 6. We notice that this time also increases almost linearly with the increasing number of input vectors. This stage is dominated by the recursive partitioning algorithm.

In order to benchmark the visualization of the histogram, each glyph is rendered with a different $16 \times 16 \times 6$ cube map texture. Figure 12 shows the frame rates that are achieved for an increasing number of glyphs. As expected, the frame rate decreases with an increasing number of glyphs. Using a modern graphics card, it is possible to render more than 1,000 glyphs at a frame rate just over 60 frames per second, which is sufficient for supporting effective user interaction. We have not tested more than 1,400 glyphs as it trends towards reintroducing visual clutter.

8 Conclusion

We have presented a technique to compute and store the distribution of three-dimensional vector directions using a cube map histogram. In addition, we use a hardware-accelerated approach to visualizing the histogram using superquadric glyphs. Our cube map histogram technique can:

• Partition the 3D vector direction space into relatively equal size histogram bins,

• Compute the bin index quickly for a given 3D vector in Cartesian space,

• Generate an accurate vector direction distribution by normalizing the bin counts with the solid angles.

We visualize the computed cube map histogram using color-textured superquadric glyphs that depict the vector field distribution of a local region. Our glyph placement strategy partitions the uniform grid space into blocks of coherent flows by using entropy and placing a glyph in each block. To allow users to freely explore the vector field, we provide an interactive visualization system. Using this system, users can interact with a treemap visualization of the partitioned space to focus on different regions of interest and change glyph density both globally and locally to utilize different levels of details. Additionally, we presented two case studies using Solar Plume and Nek dataset, and reported the performances of both the cube map computation and the visualization system.

One limitation of our space partitioning algorithm is that we sometimes cannot separate two opposite flow directions within a block, because the entropy alone cannot determine the number of peaks in the histogram. As a future work, we plan to explore using image processing algorithms to analyze the cube map histogram as six connected 2D images to discover the existence of multiple peaks. Furthermore, the performance of computing cube map histogram can be improved by using prefix sum and expanded to use on supercomputers.

References

[1] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, Jun 1991.

[2] R. Borgo, J. Kehrer, D. H. S. Chung, E. Maguire, R. S. Laramee, H. Hauser, M. Ward, and M. Chen. Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications. In M. Sbert and L. Szirmay-Kalos, editors, *Eurographics 2013 - State of the Art Reports*. The Eurographics Association, 2012.

[3] E. Boring and A. Pang. Directional flow visualization of vector fields. In *Proceedings of the 7th Conference on Visualization '96*, VIS '96, pages 389–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[4] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Press, 1999.

[5] R. Crawfis and N. Max. Direct volume visualization of three-dimensional vector fields. In *Proceedings of the 1992 Workshop on Volume Visualization*, VVS '92, pages 55–60, New York, NY, USA, 1992. ACM.

[6] D. Dovey. Vector plots for irregular grids. In *Visualization, 1995. Visualization '95. Proceedings., IEEE Conference on*, pages 248–253, 459, Oct 1995.

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[8] P. Fischer, J. Lottes, D. Pointer, and A. Siegel. Petascale algorithms for reactor hydrodynamics, 2008.

[9] N. Greene. Environment mapping and other applications of world projections. *Computer Graphics and Applications, IEEE*, 6(11):21–29, Nov 1986.

[10] M. Hlawatsch, P. Leube, W. Nowak, and D. Weiskopf. Flow radar glyphs - static visualization of unsteady flow with uncertainty. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):1949–1958, Dec 2011.

[11] B. Johnson and B. Shneiderman. Treemaps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. 2nd International Visualization Conference 1991. IEEE*, pages 284–291, 1991.

[12] G. Kindlmann. Superquadric tensor glyphs. In *Proceedings of the Sixth Joint Eurographics - IEEE TCVG Conference on Visualization*, VISSYM'04, pages 147–154, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[13] G. Kindlmann and C.-F. Westin. Diffusion tensor visualization with glyph packing. *Visualization and Computer Graphics, IEEE Transac-*

tions on, 12(5):1329–1336, Sept 2006.

[14] R. M. Kirby, H. Marmanis, and D. H. Laidlaw. Visualizing multivalued data from 2d incompressible flows using concepts from painting. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, VIS '99, pages 333–340, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[15] R. S. Laramee. First: a flexible and interactive resampling tool for cfd simulation data. *Computers & Graphics*, 27(6):905–916, 2003.

[16] P. Leopardi. A partition of the unit sphere into regions of equal area and small diameter. *Electronic Transactions on Numerical Analysis*, 25:309–327, 2006.

[17] S. Lodha, A. Pang, R. Sheehan, and C. Wittenbrink. Uflow: visualizing uncertainty in fluid flow. In *Visualization '96. Proceedings.*, pages 249–254, Oct 1996.

[18] K. Moreland. Diverging color maps for scientific visualization. In *Advances in Visual Computing*, pages 92–103, 2009.

[19] Z. Peng and R. S. Laramee. Vector glyphs for surfaces: A fast and simple glyph placement algorithm for adaptive resolution meshes. In *Proceedings of the Vision, Modeling, and Visualization Conference 2008, VMV 2008, Konstanz, Germany, October 8-10, 2008*, pages 61–70, 2008.

[20] F. Reinders, M. E. Jacobson, and F. H. Post. Skeleton graph generation for feature shape description. In W. de Leeuw and R. van Liere, editors, *Proc. Data Visualization 2000*, pages 73–82. Springer Verlag, 2000.

[21] I. A. Sadarjoen and F. H. Post. Detection, quantification, and tracking of vortices using streamline geometry. *Computers & Graphics*, 24(3):333–341, 2000.

[22] C.-F. Westin, S. E. Maier, H. Mamata, A. Nabavi, F. A. Jolesz, and R. Kikinis. Processing and visualization of diffusion tensor MRI. *Medical Image Analysis*, 6(2):93–108, 2002.

[23] C. Wittenbrink, A. Pang, and S. Lodha. Glyphs for visualizing uncertainty in vector fields. *Visualization and Computer Graphics, IEEE Transactions on*, 2(3):266–279, Sep 1996.

[24] T. Zuk, J. Downton, D. Gray, S. Carpendale, and J. Liang. Exploration of uncertainty in bidirectional vector fields. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6809, 2008. published online.