GlyphSea: Visualizing Vector Fields

Emmett McQuinn*[ab], Amit Chourasia[ba], Jürgen P. Schulze[da], Jean-Bernard Minster[ca]

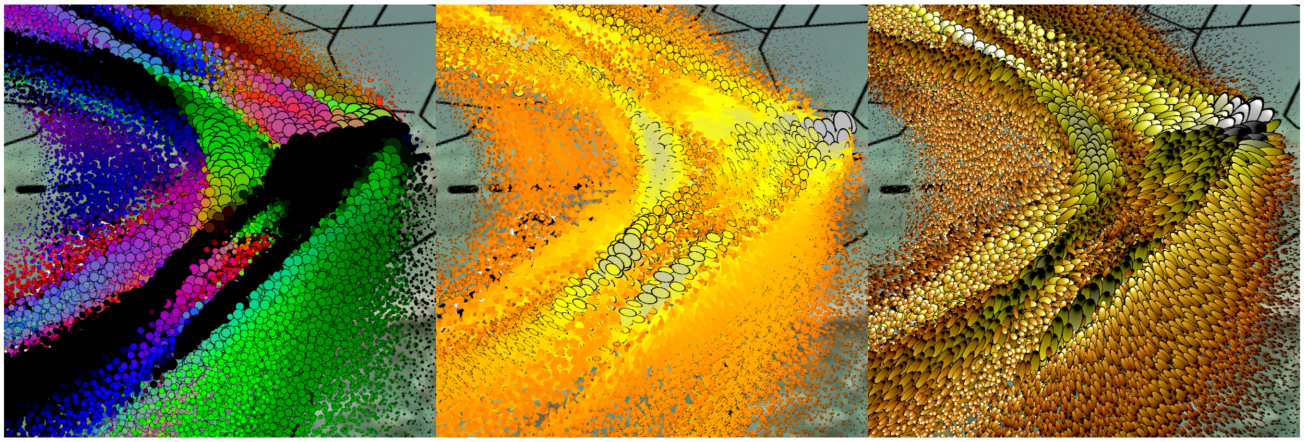
[a]University of California San Diego, 9500 Gilman Drive, La Jolla, USA; [b]San Diego Supercomputer Center, 9500 Gilman Drive, La Jolla, USA; [c]Scripps Institution of Oceanography, 9500 Gilman Drive, La Jolla, USA; [d]California Institute for Telecommunications and Information Technology, 9500 Gilman Drive, La Jolla, USA

ABSTRACT

Understanding vector fields is important in many science and engineering domains. Glyphs are often used to represent vector data as arrows, cones, ellipsoids, and other geometric shapes. When implemented using traditional 3D graphics, these glyphs have drawbacks of being view dependent, orientation ambiguous, and requiring specific geometric resolutions. We propose a straightforward new method of procedural dipole texturing of glyph shapes, which overcomes these drawbacks and can enhance existing methods. We demonstrate our method with an interactive application (GlyphSea), which incorporates additional features such as screen space ambient occlusion, glyph displacement, lattices, halos and other contextual visual cues. We also discuss the results and informal feedback from scientists on insights gained by exploring time varying vector datasets in astrophysics and seismology.

Keywords: Glyph-based Techniques, Vector Field Data, Time-varying Data, Visualization in Physical Sciences and Engineering

* work performed by Emmett at SDSC and UCSD, currently with IBM-Research Almaden. E-mail: hello@emmettmcquinn.com



(a) Existing, RGB as Vector Orientation (b) Existing, Cones with Flat Shading (c) New, Ellipsoids with Dipole Texture

Figure 1: Comparative visualization of previous glyph methods (a, b) with our novel dipole texture technique (c). This data depicts formation of a mach cone from ground velocities for a magnitude 8.0 earthquake simulation. All methods have glyph size proportional to the magnitude of ground velocity. (b) and (c) use color to show magnitude. In (a) the vector orientation is shown by mapping RGB colorspace to a normalized vector while in (b) the vector field is represented by cone glyphs. In (c) ellipsoids are used with our dipole texturing method. Our method is able to show the undulating ground motion where the white tipped ellipsoids indicate motion towards the viewer while black tipped ellipsoids indicate motion away from the viewer. This technique enables an intuitive and extensible method to visualize vector fields.

1. INTRODUCTION

Advances in computing hardware have made it possible to simulate a wide range of physical phenomena. Vector data is an important component of many science and engineering applications such as astrophysics, climate, earthquake, and turbulence simulations. Analysis of this data is crucial to understand underlying physical phenomena and for validating simulation models. One powerful and intuitive method to analyze these datasets is the use of visualization.

In this study we focus on glyphs to encode and display vector information. Glyphs offer an easy way to represent vector fields, where each glyph is positioned per field point and oriented with respect to vector direction. Color, scale, and opacity can be applied to the glyphs to represent velocity magnitude.

The two main drawbacks of glyphs are that glyphs occupy considerably large screen space to be visible and that glyphs are view dependent and ambiguous. Both the viewer's perspective and underlying data can cause ambiguous plotting with standard glyph shapes. For example, a cone or arrow glyph will look like a circle when pointing toward the viewer. An ellipsoid degenerates into a sphere when all components are equal, making it impossible to decipher vector orientation. Even with a perceptually ideal ellipsoid with a dominant major axis there is still binary ambiguity to orientation due to symmetry. These limitations mean glyphs are often constrained to a very sparse representations or scenarios where orientation ambiguity is not a concern such as in tensor visualization.

We present a quiver of realtime glyph rendering techniques that directly attack these standard glyph limitations. Our work incorporates glyphs such as comets and ellipsoids which are effective at displaying vector data while overlapping to reduce screen space requirements, a resolution independent rendering technique useful with large screens that increase screen space, and a new texturing technique to remove view and data ambiguities for common glyph shapes. The following are the key contributions of our study:

- A technique to encode and display vector data by using procedural dipole texturing, which is extendable to arbitrary glyph shapes. This method enables us to identify macro level features from far and micro level features when close.
- A method for enabling distinction of glyphs by using user tunable wire frame lattice.
- Comprehensive demonstration and discussion of glyph techniques applied to astrophysics and seismology datasets.

2. RELATED WORKS

Glyph-based rendering is extensively employed in scientific visualization. One recent use of glyphs is to represent DT-MRI tensor fields with interesting visualizations by Westin et al.[27] and Bergmann et al.[1] Nayak et al.[23] use glyphs to display discrete points on the ground surface from realtime seismic sensor data. Neeman et al.[24] applied plane-in-a-box tensor glyphs to represent stress for a single timestep of a geomechanics simulation. To our knowledge we are the first to use glyphs with full volume visualization of seismic simulation data focused on interactivity.

With the introduction of the programmable graphics pipeline on commodity hardware, it is feasible to look at volumes interactively using particle and glyph systems. Two groups simultaneously created GPU-based particle systems that reduce memory copies from a GPU to a CPU and leverages the GPU's expansive vector processing capabilities[16].[17] Stefan Gumhold[10] discovered a way to splat ellipsoids with GPU shaders. Kondratieva et al.[18] combined splatted ellipsoid glyphs with a GPU particle system.

Glyph geometry is a well studied topic, and there exist many geometries to use. We chose four basic glyph types: spheres, ellipsoids, comets, and cones. Gordon Kindlmann[13] showed that superquadratic glyphs are advantageous due to decreased perceptual ambiguity. However they increase implementation complexity, degrade interactive performance, and do not completely resolve view dependence problems. Gumhold[10] suggests a clever implementation of ellipsoid splatting which uses a simple ellipsoid intersection test by translating to spherical coordinates. Our implementation is derived from this trick. We implement a comet glyph which is similar in appearance to the one proposed by Guthe et al.[11] We chose these four glyphs due to their compactness, simplicity, familiarity, and implementation ease. While more complex glyph geometries are possible, such as an arrow, they are difficult to implement with a fast procedural method necessary for large volume visualizations.

Mesh regularity can introduce distracting visualization artifacts, such as Moiré patterns. One method used to compensate for regularity is to modify the density of glyphs in a region. This can be either done by automated means or by interactive means through use of a particle probe. Kindlmann explored the concept of glyph packing, which can compensate for both regularization artifacts and occlusion.[15] Kondrativa has also explored a particle probe for seeding glyphs[18] which mitigates distraction. Another method to compensate for regularity is using stochastic jittering, which was compared to other 2D methods in a user study by Laidlaw et al.[20] We have employed this method in our application.

Similar to our method of dipole texturing, Gordon Kindlmann[14] used a barycentric colormap transformation to provide orientation enhancements for spherical glyphs. However, the complex shading model proposed in this study makes it useful for limited cases. Our dipole texturing method provides unambiguous orientation with a procedurally generated texture that is geometry agnostic.

Various illustrative context techniques have been explored to enhance glyph geometry. Guthe et al.[12] proposed halos to distinguish glyphs from one another. Gribble et al.[9] demonstrate advanced lighting models that incorporate shadows, ambient occlusion, and diffuse inter-reflection that provide additional depth cues. Everts et al.[7] describe depth dependent halos to enhance line geometry with depth cues based on line width. We employ an edge outline to enable glyph distinction rendered with the geometry. Luft et al.[22] suggest a method of illustrate depth by unsharp masking of the depth buffer. We use Luft's method[22] to implement screen space ambient occlusion (SSAO) that is able to provide advantages of depth dependent halos[7] where the runtime is dependent on the number of imagespace pixels rather than scene geometry. This is important for realtime interaction with dense volumes comprised of a large number of glyphs.

3. GLYPH TECHNIQUES

We describe an assay of glyph techniques. First, common glyph geometries are described. Second we describe texture and shading where we introduce our procedural dipole texture method. Subsequently we describe size, opacity, and jitter for glyphs. These methods collectively provide a highly customizable interface enabling an intuitive representation of volumetric vector data.

3.1 Geometry

Glyph geometry is a core method to encode data into visual form. Instead of approximating geometry with polygons or precomputed lookup textures, we dynamically generate each glyph shape on a billboard, with the benefit of resolution independent glyphs, performance, and avoiding potentially combinatoric texture storage requirements with many appearance parameters. We explored four basic glyph geometries that can be interactively switched to explore features of interest. A brief description of glyph shapes is given as follows:

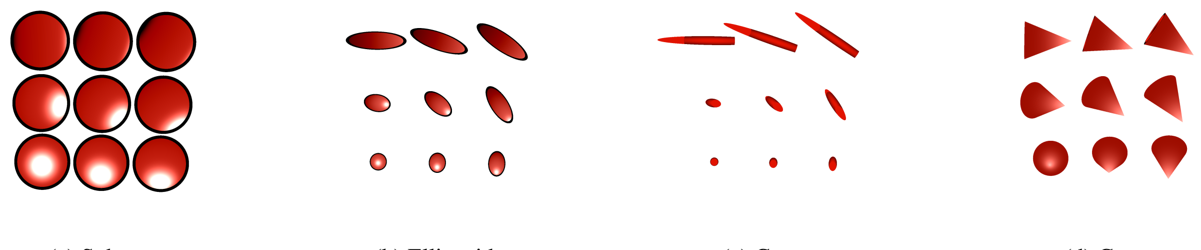

(a) Sphere (b) Ellipsoid (c) Comet (d) Cone

Figure 2: Four types of glyph geometries. (a, b, c, d) show the sphere, ellipsoid, comet, and cone glyphs respectively. All indicate vector direction using dipole texturing.

- **Sphere**: It is the simplest glyph geometry. Spheres inherently do not provide an indication of orientation due to radial symmetry (Figure 2a).
- **Ellipsoid**: This shape is created by using the velocity components as the radii for three different axes (Figure 2b). Ellipsoids require extra computation when compared to sphere glyphs.
- **Comet**: The shape of a comet is a skinny ellipsoid with fixed radii ratio and the tip removed on one end (Figure 2c). Comets require additional computation when compared to ellipsoids.

- **Cone**: The cone shape has a fixed radius to height ratio of two and is oriented with its tip heading along the vector direction (Figure 2d). The cone is a commonly used geometry by existing methods and serves as a good fit to compare and contrast methods. They require additional computation when compared to comets or ellipsoids.

3.2 Texture and Shade

Glyph texture and shade can play an important role in revealing features of interest (Figure 12). We introduce a method of dipole texturing which indicates glyph orientation in a view independent and unambiguous manner combined with halos to enhance glyph distinction. These features are created dynamically within pixel shaders rather than with traditional texture sampling. A brief description of these methods is provided below:

- **Flat Shading**: A solid color is used to shade the glyph.
- **Halos**: A dark outline is drawn around the glyph, providing distinction among neighboring glyphs in imagespace.
- **Dipole Texturing**: A new method that encodes orientation information of a vector by procedurally painting light and dark spots on opposite ends of a glyph. We exploit the human visual system's excellent acuity at detecting intensity changes.[8] The lighter white spot indicates the heading direction of the vector, whereas the dark black spot is the antipodal point indicating the tail of the vector. This technique enables the viewer to unambiguously identify the vector direction independent of the viewing angle, as the viewer can see either a light spot or a dark spot or both. The application of procedural dipole texturing to spheres enables us to encode orientation, while in case of ellipsoids and comets it reinforces orientation and removes ambiguity. Further, this technique can be extended to arbitrary glyph geometries. This method is capable of showing macro level patterns from far viewpoints and micro level features when zoomed in.

3.3 Size

Varying glyph size is an intuitive way to highlight features of importance and interest. This also helps to reduce occlusion of interior glyphs in volumetric data. We have developed interactive filters to carve out features of interest from volumetric data by defining glyph size based on a function of the vector (Figure 3). We provide a few implicit functions: linear, power, logarithmic, exponential, and uniform in order to scale the glyphs based on the underlying data. In combination with realtime temporal derivatives and integration, a vector field of velocity could scale glyphs relative to energy, acceleration, and displacement. Additionally, we provide a Bezier curve editor to sketch custom scaling functions. This is very useful allowing a scientist to remove clutter carving out features of interest in the data.

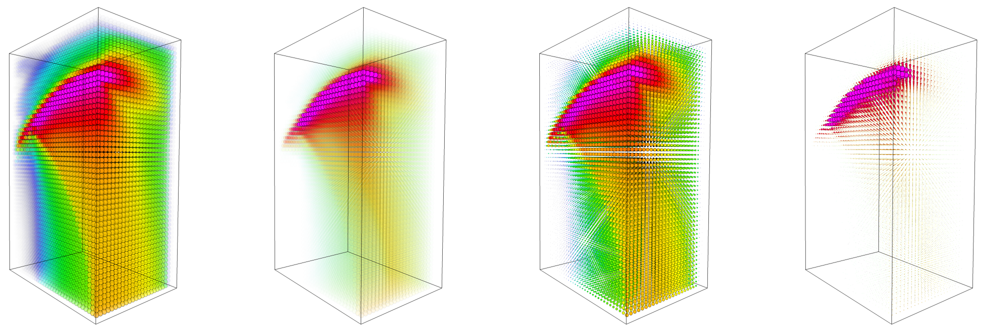

(a) Logarithmic Opacity (b) Power Opacity (c) Logarithmic Size (d) Power Size

Figure 3: Application of opacity and scale based on logarithmic and power functions which reveals the volume interior.

3.4 Opacity

Glyph opacity can also be employed to reveal the interior of a volume. We implemented similar implicit and sketch based methods used for glyph scaling to specify opacity. However, as opacity rendering is view dependent, it incurs overhead during view changes decreasing interactivity. The first row in Figure 3 shows how opacity makes it difficult to discern individually, even with halos. In contrast, scaled glyphs can be distinguished easily and are a viable alternative that do not incur a performance setback.

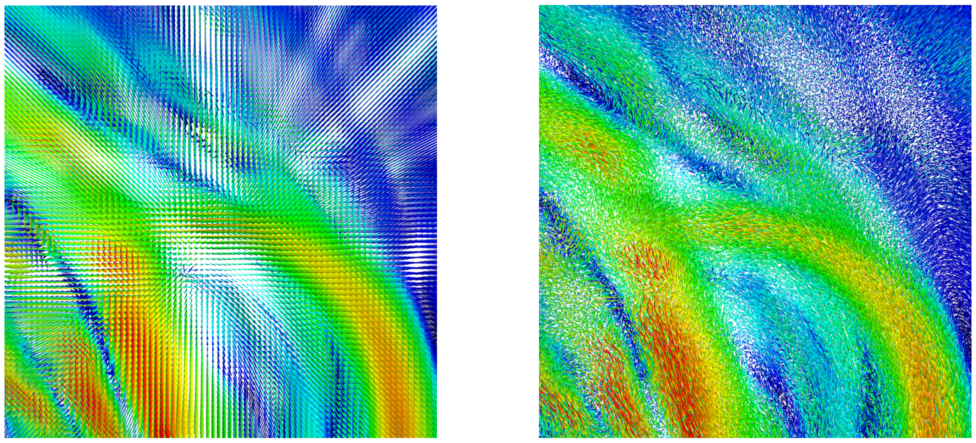

(a) No Jitter (b) Jitter

Figure 4: Mitigation of distracting patterns. (a) shows Moiré patterns caused by uniformly spaced data. The distraction caused by these patterns is mitigated by jittering the position of the glyphs by a pseudo-random amount that can be interactively tuned, as shown in (b).

3.5 Jitter

When glyphs are placed on the regular grid they create Moiré patterns (Figure 4) in perspective view, which distracts the viewer from focusing on the region of interest. This problem is further exacerbated with stereo displays. Laidlaw et al.[20] performed a user study with 2D glyphs which indicate that a jittered field improves comprehension over a regular grid. We have implemented two methods to mitigate this effect: the first method is to use jitter where we offset the glyph position with a pseudo-random amount such that the particles remain in their voxel region. This stochastic jittering method drastically reduces the Moiré patterns. In addition, jitter probabilistically reduces occlusion of glyphs stacked in columns orthogonal to the viewing plane. We provide an interface to customize jitter distance interactively. In our experience, we have found that the jitter distance of about half grid distance from the center of the glyph is sufficient to eliminate Moiré. The second method is to displace the glyph described later in the Displacement section.

4. CONTEXT TECHNIQUES

Glyphs by themselves are a useful visualization method. However, the utility of glyphs can be greatly improved by providing relevant contextual cues. We introduce a lattice to provide spatial cues for glyphs and employ screen space ambient occlusion to enhance depth perception. Further well understood techniques such as using an isosurface and texture slice provide additional context. These methods are described in the following subsections.

4.1 Screen Space Ambient Occlusion

Ambient occlusion improves depth perception and glyph distinction (Figure 5). We use screen space unsharpening of the depth buffer originally described by Luft et al.[22] since it is computationally cheap and does not require normal data from the scene to be stored in the rendering pipeline. This method can be easily added to a fixed function pipeline without significant additional overhead or re-engineering. Runtime is dependent on the number of pixels rendered regardless of the scene geometry and viewing position, thus making it beneficial for interactive systems.

However, the blur kernel radius as well as the near and far clipping planes for the depth buffer must be configured for the glyph field. We scale the blur kernel radius by the depth values such that further objects have less contribution. The choice of near and far clipping planes of the depth buffer have a significant impact on display and thus should be chosen carefully. It is desirable to redraw the depth buffer of the scene with a custom near and far clipping plane just for the occlusion pass. Additionally, we exaggerate and clamp the occlusion factor to highlight objects in front (Figure 5).

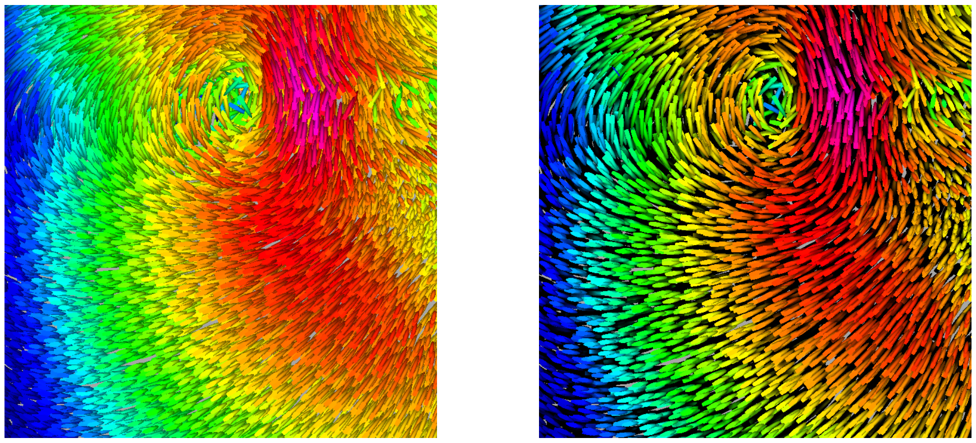

(a) SSAO off (b) SSAO on

Figure 5: Comparison of the comet glyph with (b) and without (a) screen space ambient occlusion.

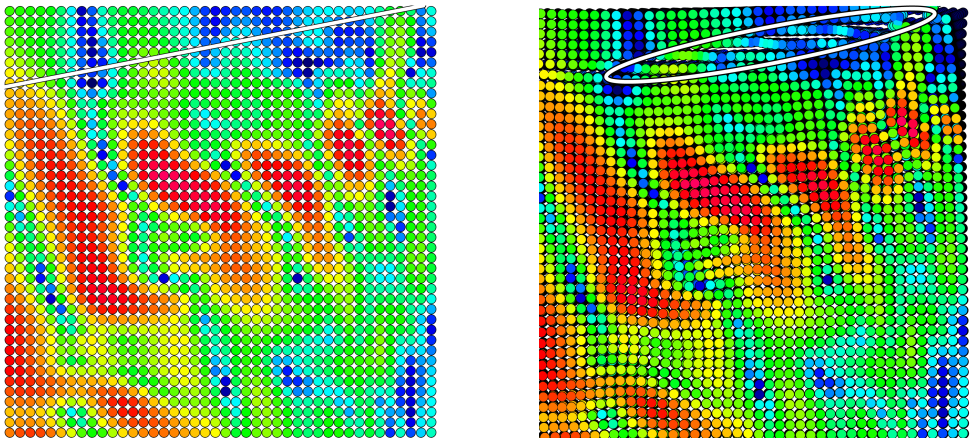

(a) No Displacement (b) Displacement 10x

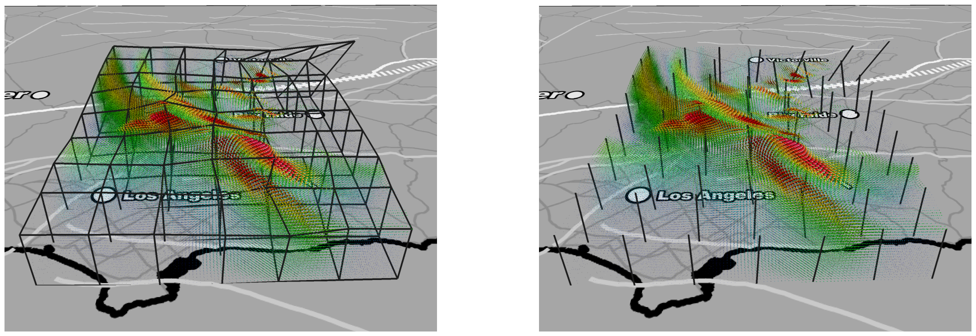
Figure 6: Application of displacement. (a) shows the result when the glyphs are in their original grid location, with the slanted white line on top indicating the earthquake fault. (b) shows the glyphs displaced by 10 times the actual displacement which reveals the shears in the fault marked by a circle).

4.2 Displacement

For velocity fields, the displaced position of glyphs can be used as an additional context for the underlying data. The glyph displacements can either be based on the scalar magnitude of vector data or can be computed from the difference of subsequent timesteps for time varying data. The movement of glyphs are a visual cue which can be utilized to reinforce and uncover interesting features. Displacement also helps to alleviate Moiré patterns on regular grid data. We provide a user tunable parameter in our application to specify displacement exaggeration.

4.3 Lattice

It can be hard to identify the spatial location of glyphs in a volumetric field. While ambient occlusion can help with depth perception, it is only useful as a depth cue. To alleviate the spatial perception problem of glyphs, we create “full-lattice” (Figure 7a) or “kelp-lattice” (Figure 7b) guides to provide context. The full-lattice displays a wireframe grid where neighboring glyphs are interconnected by lines. This gives a strong sense of neighborhood, but it also introduces additional clutter. Inspired by the undersea kelp forest, we created a kelp-lattice where the connecting lines are perpendicular to the ground surface, which helps to reduce clutter. A side effect of employing either lattice is that the glyph size should be small to reduce visual clutter and prevent occlusion of the lattice and volume in general. To minimize clutter we provide an interface to customize the number of grid lines that are shown. The glyph displacement deforms the lattice’s wireframe guides based on glyph location, again helping to reinforce the neighborhood context of the glyphs.



(a) Full-Lattice (b) Kelp-Lattice

Figure 7: Results of lattice displacements. It is often difficult to associate neighbors of glyphs. To overcome this, we created a wireframe lattice as shown in (b). However the lattice adds more clutter to an already dense volume. Instead, we only show kelp-like vertical lines (b), reducing visual clutter while retaining neighborhood information.

4.4 Isosurface and Slice

We have implemented isosurfaces using the Marching Cubes algorithm.[21] This additional context can help with the exploration of data, with an example described in the TeraShake use case (Figure 9).

Slices are used to show scalar contextual information in conjunction with glyphs. A textured volume slice is used to display scalar information of the volume. This is helpful for the scientist to investigate vector magnitudes at an arbitrary volume depth.

5. IMPLEMENTATION

The main loop of our engine performs three phases: loading the data, computing the glyph attributes, and updating the glyphs' display. The system has two main threads, one for loading data, and the other for rendering. Decoupling the rendering loop from the data manipulation is important in keeping interactivity high.

We chose to update glyph positions within the CPU, rather than GPU, because of the diverse nature of our input data requiring streaming data from disk. Each glyph has an orientation, color, radius, and position which can be independent in some datasets and thus one cannot simply compute all components on the fly. As an optimization we perform state checks to reduce copies to and from the device. This ensures that data is only copied to the GPU when values have changed. Additionally, our code is written to have high cache utilization by keeping data packed and having reduced loop complexity only accessing one array type at a time. As a benefit, compilers like *GCC* and *ICC* can perform autovectorization for these simple loops.

The engine is implemented in C++ using the OpenSceneGraph[*] library with custom GLSL and OpenGL 2 code for drawing glyphs. This has enabled porting our application to all major operating systems (Linux, Mac, and Windows) and several high resolution display systems. Glyph state is not updated with the traditional OpenSceneGraph state operations, but rather modified via direct array modifications. These arrays are then copied to their respective OpenGL arrays. This is similar to how animated textures are implemented with OpenSceneGraph. Future efforts may reduce data transfer with compression and interpolation techniques inside the GPU.

We implemented screen space ambient occlusion with a post effects pipeline. The scene is rendered to a camera with an offscreen framebuffer. This framebuffer is then mapped to a screen aligned quad. This quad then passes through an x-blur and then a y-blur of the depth buffer, and then an occlusion pass contributes shadowing to the scene. The final pass in the post effect pipeline renders a texture of the offscreen colorbuffer.

[*]http://www.openscenegraph.org/

5.1 Glyph Field Implementation

Our implementation uses billboarded glyphs rendered with GLSL shaders leveraging the programmable graphics pipeline. The data is submitted as points with orientation captured in the normal vector, color as a 4-component vector, radius as a 1 component texture coordinate, and position as a 3-component vector. This data passes through a vertex shader to a geometry shader, which creates a screen-aligned billboard with correct texture coordinates. It also precomputes the glyph rotation, but this varies for each type of glyph geometry. Finally, the fragment shader uses raycasting to determine geometry intersection discarding unused fragments and coloring glyphs.

There are four geometries used: sphere, ellipsoid, comet, and cone. All use raycasting techniques which calculate ray-geometry intersection, discarding the rays which are invalid and do not hit geometry. Once the point of intersection is known, then the fragment is colored based on the shading model desired.

The geometry computations here are not perspectively correct. This would require additional steps such as those described by Gumhold in.[10] Instead, we employ an orthographic simplification that relies on the fact that glyphs are locally orthogonal but globally the centers are perspectively correct. Any perspective distortion is minimized when glyphs are small. The scale of the billboard is perspectively correct, so that glyphs can have small scale either when far away or simply have a small radius. With vector fields such as those depicted in this paper, perspective distortion does not appear to be an issue. Using an orthographic projection reduces the data passed and the computation necessary through the programmable graphics pipeline, where otherwise one may have interpolator bound performance on older graphics cards.

In addition, the glyphs represented here do not have exact depth. Calculating depth is not a complex extension of the current glyphs, but simply calculating and setting the correct depth stops early-z culling in the graphics pipeline resulting in non-interactive performance. As long as glyphs do not occupy the same volume of space, this is not a concern. We found that when glyphs do intersect, such as in Figure 5, it was not visually distracting enough to motivate immediate study.

5.2 Sphere Geometry

Spheres require a very simple intersection test. Because the billboard is aligned with the camera, the texture coordinates are linearly related to imagespace coordinates. Texture coordinates are typically specified in the range $[0,1]$ but are useful with glyph rendering to have an origin in the center of the texture such that the new range is $[-1,1]$. This is done in a geometry shader. Given texture coordinates $\vec{p}$, a sphere with radius r (in terms of texture coordinates), and a sphere with center $\vec{c} = \langle 0,0 \rangle$, the intersection test is the following:

$$||\vec{c} - \vec{p}|| \leq r$$

This condition holds where the imagespace location is within the sphere volume. Intersections tests for the sphere, ellipsoid, and comets happen within the pixel shader and are fairly cheap. When the intersection test fails, the pixel is discarded and rendering continues to other primitives.

Surface normals are necessary in order to provide useful texturing and shading of surfaces. For a sphere, the normals can be calculated very simply. If a ray intersects with surface location $\vec{p}$ and the sphere has origin $\vec{o}$, In model view coordinates, the sphere is centered around the origin $\vec{o} = \langle 0,0,0 \rangle$, resulting in the simplification for the surface normal:

$$\vec{n} = \frac{\vec{p} - \vec{o}}{||\vec{p} - \vec{o}||}, \;\; \vec{o} = \langle 0,0,0 \rangle \quad \rightarrow \quad \vec{n} = \frac{\vec{p}}{||\vec{p}||}$$

The dipole technique is rendered using a normal that is rotated to match the orientation of the glyph. Given the vector direction $\vec{v}$ and surface normal $\vec{n}$, dipole texturing is calculated for two different colors for two antipodal points. Both antipodal points share similar lighting equations, with a diffuse-like component I_d, specular-like component I_s, and dipole attenuation factor s:

$$I_d = \vec{v} \cdot \vec{n} \quad I_s = {I_d}^s$$

Diffuse-like intensity I_d provides a subtle effect that enhances the exaggerated specular-like intensity I_s. A test is created to determine which antipodal point is being rendered: $sign = \vec{v} \cdot \vec{n}$. If $sign$ is less than zero, then the pixels being rendered are on the "dark" side of the glyph. When this is the case, then both I_d and I_s are set to negative values. Due to color clamping, a negative value will bring the color to $rgb = \langle 0,0,0 \rangle$ which represents black for the dark side of the glyph. The final color is a blending of the dipole color and the ambient color:

$$color_{out} = K_a * color_{in} + K_d * I_d + K_s * I_s$$

where typically $K_a + K_s > 1$ to provide strong dipole distinction. Current parameters were selected by trial and error to determine a good blending with the values depending on glyph geometry. For the ellipsoid glyph, $s = 100$, $K_a = 0.7$, $K_d = 0.4$, and $K_s = 0.6$. The sphere glyph has slightly different values: $s = 10$, $K_a = 0.8$, $K_d = 0.3$, and $K_s = 0.8$.

5.3 Ellipsoid Geometry

Ellipsoids are drawn with a technique similar to that described by Gumhold in,[10] which leverages the simplified axis-aligned ellipsoid ray-intersection test to reduce computation. The intersection test for an axis-aligned ellipsoid centered around the origin can be described with the following equation:

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 \leq r^2$$

where r_x, r_y, and r_z are the radius for the principal axes of the ellipsoid, and r is a uniform scale factor for the ellipsoid.

This intersection test can be simplified due to how the scene is structured. If the ellipsoid is placed axis-aligned and centered about the origin, then camera rays will all be $\vec{v} = \langle 0,0,1 \rangle$. This means that to test for intersection, the two relevant components are x and y, such that one can effectively set z to zero for the case of detecting intersection. This equation is cheap to compute and efficient with vector hardware.

This method is not yet sufficient for glyphs, because rotations cause ellipsoids to not be axis aligned. Instead, one can project the ellipsoid to a sphere and apply the simple ray-sphere intersection test in this new space. Some rotation R provides an orthonormal basis for the ellipsoid axes which may not align with the camera's axes. GlyphSea constructs a rotation matrix using the concept that in the current implementation the axes have fixed lengths and the principal axis corresponds with the x-axis $\vec{x}$ before projection. After projection, this principal axis needs to be oriented with $\vec{d}$. This method works because the non-major axes of the ellipsoid have equal length and thus have rotational symmetry about the major axes. This would not be ideal if a texture or glyph did not have rotational symmetry. This rotation matrix is constructed within the geometry shader.

An ellipsoid is considered "axis aligned" when the principal axes project to the x, y, and z axes. The math for calculating ray-ellipsoid intersection is simpler when an ellipsoid is axis aligned. Within the pixel shader, the texture coordinate positions $\vec{t}$ are rotated to produce an axis aligned ellipsoid, mapped from an axis aligned ellipsoid to an axis aligned sphere, and then inverse the rotation to provide a non-axis aligned sphere test [†]. This is performed with the following steps where the ellipsoid axes are of lengths $\langle r_x, r_y, r_z \rangle$:

$$\mathrm{S} = \begin{bmatrix} 1/r_x & 0 & 0 \\ 0 & 1/r_y & 0 \\ 0 & 0 & 1/r_z \end{bmatrix}$$

$$\vec{t'} = \mathrm{R^T S R}\vec{t}$$

Testing for intersection is now simply testing for sphere intersection. This is accomplished with the familiar test $\vec{t'}^2_x + \vec{t'}^2_y \leq r^2$ that looks for the intersection point of a sphere with the plane $z = 0$.

The dipole is rendered in the same manner as described with sphere geometry, where a normal is calculated and rotated to axis-aligned coordinates.

[†]Because the ellipsoid is splatted on an axis-aligned billboard, texture space and screen space coordinates are constructed to be identical with some offset and scaling factor.

5.4 Comet Geometry

Comets are actually very thin ellipsoids with a cap removed. Removing the cap is performed by testing the length along the principal axis, and clipping a certain radius away from the origin of the ellipsoid. This is done by rotating the principal axis so that it is axis aligned, and then measuring the distance along this line. If the distance is greater than 0, it is on one half of the ellipsoid. Then a refinement step determines if the distance to the texture coordinate is more than a small radius to create a round head.

Using the parameters described in the ellipsoid section, distance along the principal axis is calculated as $d = (\mathrm{R}\vec{t})_x$. This can be stored in the fragment shader with the original ray intersection tests rather than requiring an additional matrix-vector multiplication. If $d > 0$, this is the positive side of the ellipsoid's principal axis (the direction it is pointing). When this is the case, the pixel shader determines if a glyph is visible if $\vec{t}_x^2 + \vec{t}_y^2 > 0.155$. Trial and error determined the value 0.155 that creates glyphs which are aesthetically pleasing and provide strong directional information.

The dipole is rendered the same manner as ellipsoid geometry, except that when $d > 0$ the comet uses flat shading.

5.5 Cone Geometry

Like ellipsoids, the cone equation is simpler using axis-aligned coordinates, where the principal direction of the cone is along the y-axis. Because the billboard is screen-aligned and we use an orthographic simplification, the camera position can be assumed to be $\langle tx, ty, 0.0\rangle$, where tx and ty are billboard texture coordinates. The camera ray is simply $\langle 0, 0, 1\rangle$. The camera ray and camera center are then rotated from to be in axis-aligned coordinates. This rotation is computed by rotating the direction vector to the y-axis. The calculation for ray-cone intersection solves the positive result of the following quadratic equation:

$$(r_x^2 + r_z^2 - c^2 r_y^2)t^2 + (p_x v_x + p_z v_z - c^2 p_y v_y)2t + p_x^2 + p_z^2 - c^2 p_y^2 = 0$$

where t is the distance along the ray $\vec{r}$ projected from the camera with center $\vec{p}$. The scalar c is a scaling factor for cone radius. Because this calculates the intersection with an infinite cone, the cone must be further clipped. Values with an intersection point above 0 are discarded, and values less than the negative height of the cone are discarded.

Drawing an endcap for the cone requires a ray-plane intersection which checks if the intersection is less than the cone radius for the base. If doing a minimum depth test one calculates the depth for both intersections and choses the smaller value. An alternative method is to calculate the ray-cap intersection, and then perform backface culling. If the plane is not culled then the cap is drawn. This test can be performed before the cone equation, to determine if the more expensive ray-cone equation is necessary.

With dipole texturing, it is not necessary to calculate exact normals for the cone nor is it usable as a pole in the case of a plane. Instead, a distance function from the center is used as a parameter to a falloff function. In the case of the base, the color is mixed from black to the base color with a linear falloff. With cone intersection, the color is composed with a blend of 70% flat, 40% $radius^3$, and 30% $radius^2$ functions. This sums to over 100% to cause the brightness to saturate.

6. PERFORMANCE AND BENCHMARKS

Our implementation uses shaders to create glyph geometry procedurally. A key benefit of using procedural glyph geometry is resolution independence. This means there are are no artifacts drawing both large or small glyphs, regardless of whether drawing on a workstation or high resolution tile display. Procedural rendering has the additional benefit of not being memory intensive for high resolution geometry with several appearance attributes, with procedural rendering submitting only one vertex per glyph to the GPU. Figure 8 shows performance in Linux with a static scene on an Intel Core i7 970 3.2GHz system with DDR3-1600 memory and a GeForce GTX 470 with 1280MB of VRAM. The graphics loop is decoupled from loading data in separate threads, which allows for interactivity while data is still loading from disk. The datasets used for this paper contain hundreds of thousands of glyphs and maintain performance above 60fps with this system. In contrast, a system using a traditional geometry based approach, such as in[25] and mirrored with our own early experiments, can only support several thousand glyphs with interactive performance.

With 950000 glyphs, performance is not interactive for ellipsoids and cones with radius 3.0, having a framerate of 4.80 and 7.96 frames per second respectively. The system becomes fillrate bound due to the large number of glyph-pixels drawn. However, when the radius is reduced to 1.0, the glyphs are interactive, with the cones having a performance of 50.40 frames per second and the ellipsoids 79.45 frames per second.

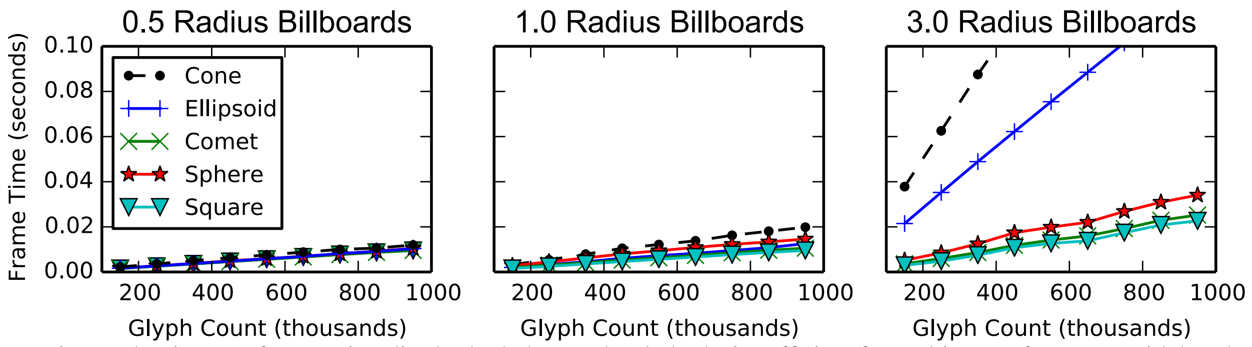


Figure 8: Render time per frame using dipole shaded procedural glyphs is sufficient for realtime performance with hundreds of thousands of glyphs. The graph shows a linear trend in performance as the number of glyphs grow in number, and a nonlinear trend as the radius increases when pixel fill rate increases. The “square” glyph is simply a flat-shaded quad, implemented with a geometry shader representing the overhead with the current system.

7. APPLICATION DISCUSSION

We used our system to visualize two different datasets, one from an astrophysics simulation and the other from a geophysics simulation. The exploration of the datasets was carried out with scientists several times during the course of application development and their input was incorporated iteratively. In each exploration session the application was driven by a visualization expert and a scientist provided guidance on where and what to explore by refining the entire array of parameters including geometry, texture, glyph scale, displacement, lattice density and thickness, and context cues such as the slice, isosurface, base map, halos, and lattice. The system displays temporal data at a user provided rate, allowing a scientist to use standard play, pause, forward, and reverse controls to move to a desired time step. Most of the exploration sessions were conducted on a PC workstation and a few were conducted in an immersive StarCAVE environment[5] and OptiPortal tile display.[6] Descriptions of the two applications are provided in the subsequent subsections.

7.1 Seismology Data

We use the publicly available TeraShake 2.1 dataset made available through the 2006 IEEE Visualization Contest. This data consists of a time varying velocity field for a simulation of a hypothetical 7.7 magnitude earthquake on the southern San Andreas fault spanning a region of 600km x 300 km x 80km mapped to a uniform grid. Seismic visualization is a moderately studied topic, with the bulk of existing work centered on visualization of scalar fields using direct volume rendering.[2–4, 26]

We worked with three seismologists for informal evaluation and feedback of our techniques. We interactively explored a cropped region of this data within 100 x 125 x 20 voxels in the Los Angeles basin which is highly interesting to scientists. We identified that contextual cues are very important to aid the understanding of ground motions. In our GlyphSea implementation we provide a mechanism to add a base map, fault plane (Figure 9), as well as ground characteristics data which describes the stiffness of the ground. In this rich context the seismologist can explore the ground velocity fields interactively. Tuning of various glyph parameters such as shape, color, texture, scale, jitter, lattice, displacement, and isosurface allows the seismologist to visualize the ground wave propagation.

Our dipole texturing method enabled scientists to easily identify the P-waves and S-waves[‡] for ground motion. Moreover the vibration like motion of ground is intuitively visible as flipping of high and low intensity dipole texture dots from a far away viewpoint (Figure 10). Scientists identified source directivity and wave guide effects which channel much of the energy towards Los Angeles away from the San Andreas fault (Figure 9). They were able to explore the wave amplification seen in the Los Angeles basin in context of a ground stiffness isosurface and also able to study secondary effects such as reflections. The scientists have also spotted previously unseen vortices (Figure 5 b) in the velocity field data, which we confirmed by computing and visualizing curl and divergence of the data. Exaggerated displacements drawn with a lattice further reinforced the ground motion direction and magnitude, but requires care as the lattice can add clutter. The scientists favored ellipsoid and comet glyphs with dipole texturing which provides an intuitive sense of ground motion direction.

[‡]P-waves or primary waves are longitudinal or compressional waves and can travel through gases, liquids, and solids. S-waves or secondary waves are transverse or shear waves, but slower than a P-wave, and can move through solids but not through gases or liquids.

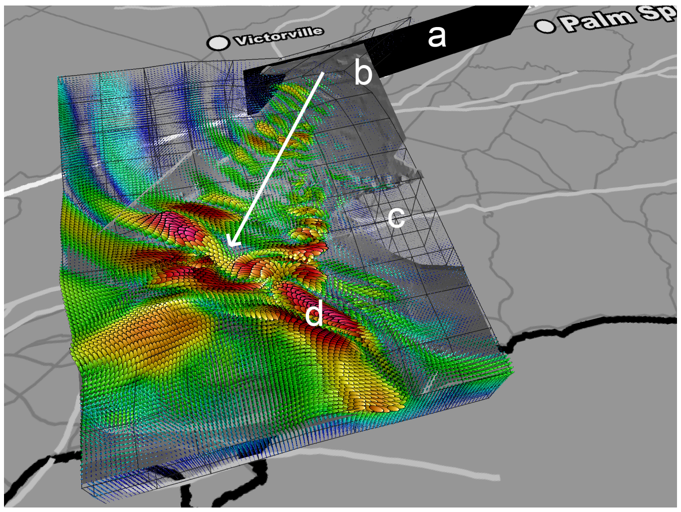


Figure 9: Visualization of Terashake data with contextual elements that help scientists identify regions of interest and seismic features. The background is a contextual map showing freeways (dark gray), fault lines (light gray) and city labels. Label (a) marks the San Andreas rupture fault plane, label (b) shows an isosurface of ground stiffness in gray color, label (c) shows the deformed lattice from exaggerated ground displacements, label (d) shows wave amplification in the Los Angeles basins. The white arrow shows the wave guide effect which channels energy away from the fault towards Los Angeles.

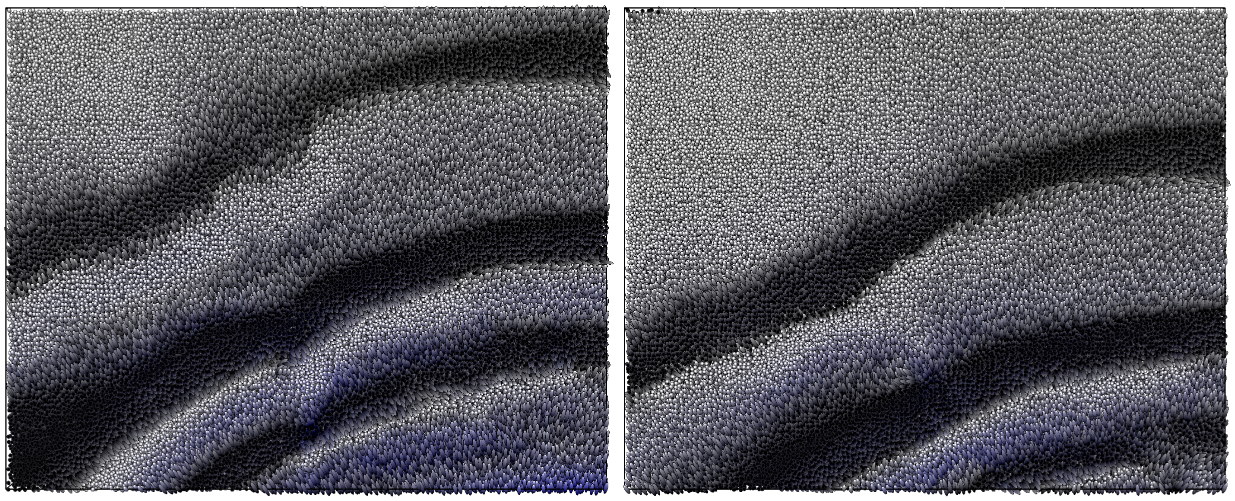


Figure 10: Snapshots of TeraShake data at two different timesteps showing ground motion P-waves with dipole texture applied to ellipsoids. Despite no distinction with individual glyphs, the macro level ground velocity direction and magnitudes are visible through the use of dipole texturing. The white tipped glyphs show motion towards the viewer while the black tipped ones show motion away from the viewer, while color depicts magnitude.

7.2 Astrophysics Data

In 3D ideal Magnetohydrodynamics (MHD), quadratic quantities such as total energy, magnetic helicity, and cross-helicity are exactly conserved. Moreover, magnetic field lines are "frozen" into the gas flow and the topology of the magnetic field configuration cannot be changed in ideal evolution. Thus, many important physical processes, such as magnetic field line reconnection, local gas heating, and particle acceleration, are disallowed. In non-ideal MHD, such processes can occur within thin layers, which are regions of sharp spatial gradients in the current density and vorticity. The tendency for formation of current and vortex singularities in ideal MHD, if and when it occurs, is a phenomenon of great interest because the sites of singularity formation are precisely the sites where important astrophysical processes such as heating and particle acceleration can occur. In the context of star formation the system is a magnetized, highly compressible turbulent interstellar medium characterized by extremely high Reynolds and magnetic Prandtl numbers. The formation

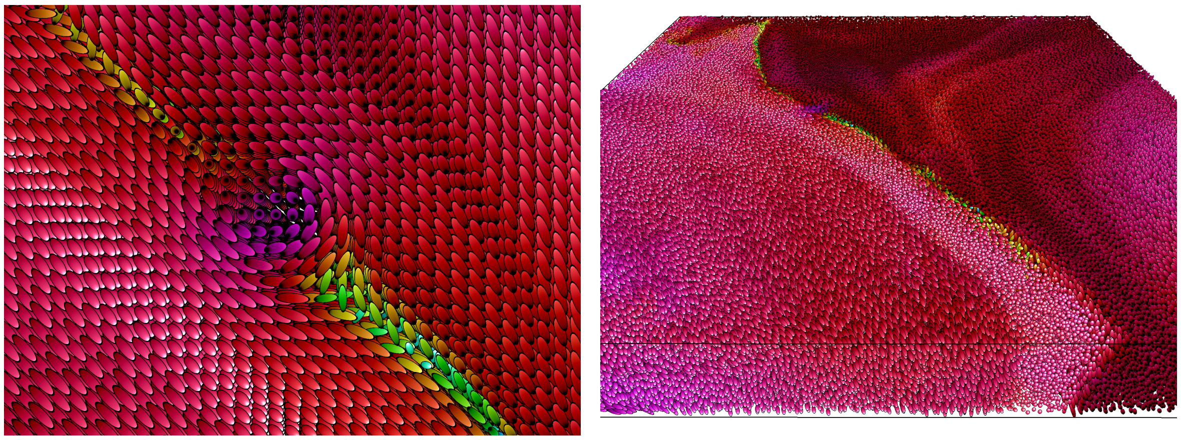


Figure 11: Left: Snapshot showing a magenta colored vortex in a magnetic field which breaks the green-yellow current sheet in half. Right: Visualization of magnetic field from a MHD simulation which shows a thin green-yellow current sheet. The dipole texture method is able to show opposing orientation of magnetic field left and right of current sheet.

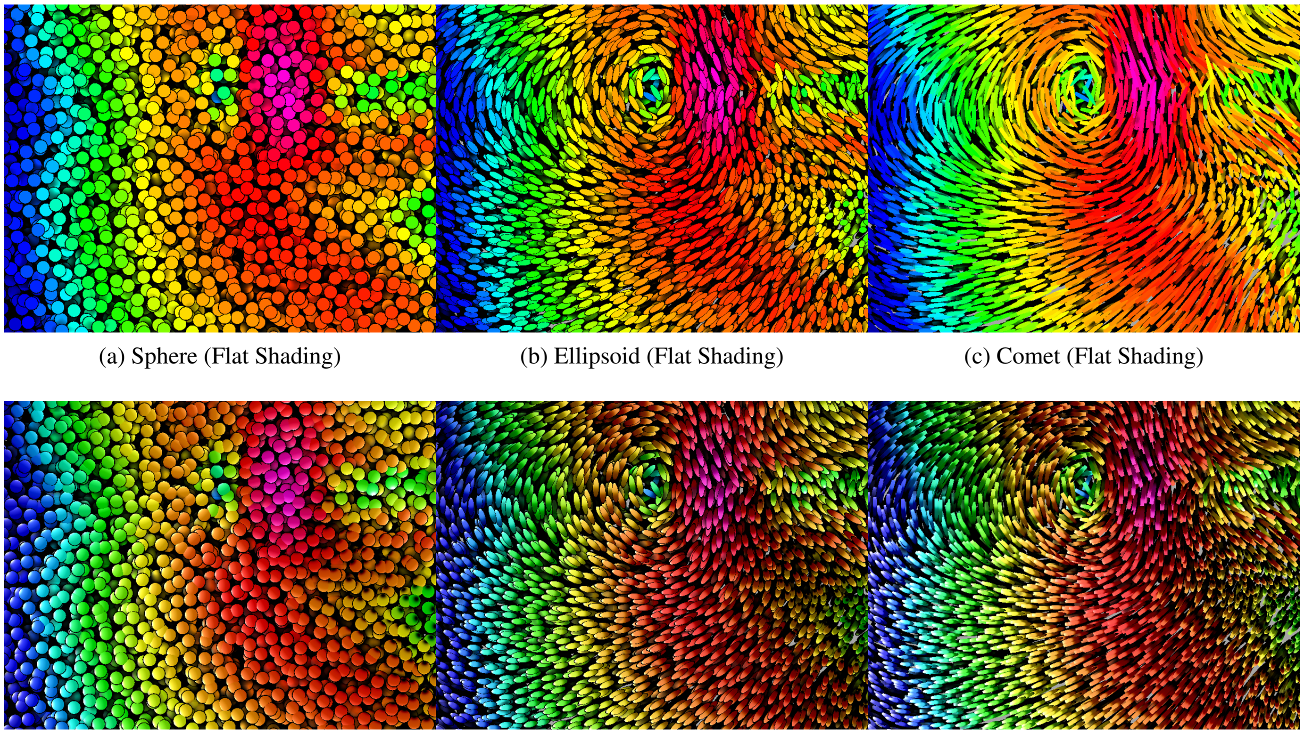
of singular dissipative structures, such as current sheets, in the context of compressible MHD turbulence is largely an unexplored territory.

We had informal exploration and discussion sessions with an astrophysicist who provided us with data.[19] We learned that his typical method for visualization was to create projected scalar volume renderings which provide only a crude level of insight. The purpose of this exploration was to see if the visualization of a vector magnetic field would allow uncovering some of the above mentioned phenomena of magnetized turbulence. During interactive visualization of the volumetric magnetic vector field data set with 150 x 150 x 18 voxels the scientist was able to spot the thin green-yellow current sheet (Figure 11) very easily as would be the case with other standard techniques. Furthermore, an “aha moment” was witnessed when the scientist was able to identify magnetic fields in opposing directions adjacent to the current sheet with our dipole texturing of glyphs (Figure 11). Another perplexing discovery was where a magnetic vortex breaks the current sheet in half (Figure 11). We found that the ellipsoid and comet glyphs with dipole texturing were most insightful. These visualizations have enabled the scientist to explore the formation mechanisms and structure of these enigmatic objects.

8. CONCLUSION

In this study we have presented a method of dipole texturing that provides a view independent and unambiguous way to encode orientation of vector data. We posit that this method is general and is extensible to arbitrary glyph shapes. We demonstrated an efficient depth enhancing method using screen space ambient occlusion which is independent of scene geometry and does not degrade performance as a function of the number of glyphs. We demonstrated dynamic glyph and texture generation which provide resolution independent per-pixel 3D information as opposed to pre-computed texture lookups. We introduced a method using the kelp-lattice and full-lattice to enhance perception of spatial neighborhoods with glyphs. The application also incorporated rich contextual multi-field information such as maps, slices, and isosurfaces that allow for a rich environment for interactive exploration. Overall the techniques implemented provides an extensive array of customizable parameters to enable rich and interactive exploration of vector data. The dipole texturing has proven to be succinct, unambiguous, and effective to display vector data and could be used as a general technique in visualization. We have demonstrated our glyph techniques with two different data sets from astrophysics and seismology. While working with the scientists we have discovered previously unseen features, who had a very positive opinion of this work.

While we have found that our techniques have uncovered insightful features they do have a few limitations which needs to be kept in mind. The dipole texturing method reserves white and black color markings to communicate orientation, thus these colors should be avoided in a color map. Glyphs inherently occupy a large screen space increases the likelihood of occlusion in volumetric datasets. This can be mitigated to some extent using high resolution displays, such as Tile Display, CAVE, or high density displays.



(a) Sphere (Flat Shading) (b) Ellipsoid (Flat Shading) (c) Comet (Flat Shading)

(d) Sphere (Dipole Texturing) (e) Ellipsoid (Dipole Texturing) (f) Comet (Dipole Texturing)

Figure 12: Glyph comparison highlighting the utility of dipole texturing. The top row has flat shading while the bottom row has dipole texturing for spheres, ellipsoids, and comets. Even if a scientist assumes that a vortex exists, the direction is ambiguous without dipole texturing (bottom row). All glyphs are uniformly scaled and rendered with SSAO.

ACKNOWLEDGMENTS

We would like to thank and acknowledge Geoffrey Ely, Steve Day, Kim Olsen, and Alexei Kritsuk for their insight, suggestions and providing data. This work was funded through National Science Foundation's OCI-0636438, EAR-0744493, SCI-0503697, OCI-0503944, OCI-0932251 awards.

REFERENCES

[1] Ø. Bergmann, G. Kindlmann, A. Lundervold, and C.-F. Westin. Diffusion k-tensor estimation from q-ball imaging using discretized principal axes. In *Nineth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'06)*, Lecture Notes in Computer Science 4191, pages 268–275, Copenhagen, Denmark, October 2006.

[2] A. Chourasia, S. Cutchin, Y. Cui, R. W. Moore, K. Olsen, S. M. Day, J. B. Minster, P. Maechling, and T. H. Jordan. Visual insights into high-resolution earthquake simulations. *IEEE Computer Graphics and Applications*, 27(5):28–34, 2007.

[3] A. Chourasia, S. M. Cutchin, K. B. Olsen, B. Minster, S. Day, Y. Cui, P. Maechling, R. Moore, and T. Jordan. Visualizing the ground motions of the 1906 san francisco earthquake. *Computers and Geosciences*, 34:1798–1805, 2008.

[4] Y. Cui, K. B. Olsen, T. H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. K. Panda, A. Chourasia, J. Levesque, S. M. Day, and P. Maechling. Scalable earthquake simulation on petascale supercomputers. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–20, Washington, DC, USA, 2010. IEEE Computer Society.

[5] T. A. DeFanti, G. Dawe, D. J. Sandin, J. P. Schulze, P. Otto, J. Girado, F. Kuester, L. Smarr, and R. Rao. The starcave, a third-generation cave and virtual reality optiportal. *Future Gener. Comput. Syst.*, 25(2):169–178, 2009.

[6] T. A. DeFanti, J. Leigh, L. Renambot, B. Jeong, A. Verlo, L. Long, M. Brown, D. J. Sandin, V. Vishwanath, Q. Liu, M. J. Katz, P. Papadopoulos, J. P. Keefe, G. R. Hidley, G. L. Dawe, I. Kaufman, B. Glogowski, K.-U. Doerr, R. Singh, J. Girado, J. P. Schulze, F. Kuester, and L. Smarr. The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Gener. Comput. Syst.*, 25(2):114–123, 2009.

[7] M. H. Everts, H. Bekker, J. B. T. M. Roerdink, and T. Isenberg. Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1299–1306, 2009.

[8] C. H. Graham and C. Cook. Visual acuity as a function of intensity and exposure-time. *The American Journal of Psychology*, 49-4:654–661, 1937.

[9] C. P. Gribble and S. G. Parker. Enhancing interactive particle visualization with advanced shading models. In *APGV '06: Proceedings of the 3rd symposium on Applied perception in graphics and visualization*, pages 111–118, New York, NY, USA, 2006. ACM.

[10] S. Gumhold. Splatting illuminated ellipsoids with depth correction. In *VMV*, pages 245–252, 2003.

[11] S. Guthe, S. Gumhold, and W. Straßer. Texture particles: Interactive visualization of volumetric vector fields, 2001.

[12] S. Guthe, S. Gumhold, and W. Straßer. Interactive visualization of volumetric vector fields using texture based particles. In *Journal of WSCG*, pages 33–41, 2002.

[13] G. Kindlmann. Superquadric tensor glyphs. In *Proceedings of IEEE TVCG/EG Symposium on Visualization 2004*, pages 147–154, May 2004.

[14] G. Kindlmann and D. Weinstein. Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 183–189, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[15] G. Kindlmann and C.-F. Westin. Diffusion tensor visualization with glyph packing. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006)*, 12(5):1329–1335, September-October 2006.

[16] P. Kipfer, M. Segal, and R. Westermann. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM.

[17] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 123–131, New York, NY, USA, 2004. ACM.

[18] P. Kondratieva, J. Kruger, and R. Westermann. The application of gpu particle tracing to diffusion tensor field visualization. *Visualization Conference, IEEE*, 0:10, 2005.

[19] A. Kritsuk, S. D. Ustyugov, and M. L. Norman. Interstellar turbulence and star formation. In *Proceedings IAU Symposium*, volume 270, pages 119–126. International Astronomical Union, 2010.

[20] D. H. Laidlaw, R. M. Kirby, C. D. Jackson, J. S. Davidson, T. S. Miller, M. da Silva, W. H. Warren, and M. J. Tarr. Comparing 2d vector field visualization methods: A user study. *IEEE Transactions on Visualization and Computer Graphics*, 11:59–70, 2005.

[21] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Aug. 1987.

[22] T. Luft, C. Colditz, and O. Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics*, 25(3):1206–1213, jul 2006.

[23] A. M. Nayak, K. Lindquist, D. Kilb, R. Newman, F. Vernon, J. Leigh, A. Johnson, and L. Renambot. Using 3D Glyph Visualization to Explore Real-time Seismic Data on Immersive and High-resolution Display Systems. *AGU Fall Meeting Abstracts*, pages C1208+, Dec. 2003.

[24] A. Neeman, B. Jeremic, and A. Pang. Visualizing tensor fields in geomechanics. *Visualization Conference, IEEE*, 0:5, 2005.

[25] T. Schultz and G. Kindlmann. Superquadric glyphs for symmetric second-order tensors. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1595–1604, 2010.

[26] T. Tu, H. Yu, L. Ramirez-guzman, J. Bielak, O. Ghattas, K. liu Ma, and D. R. O'hallaron. From mesh generation to scientific visualization: An end-to-end approach. In *in SC2006*, 2006.

[27] 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.