Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

Interaction of Light and Tensor Fields

Xiaoqiang Zheng and Alex Pang

Computer Science Department
University of California, Santa Cruz, CA 95064
zhengxq@cse.ucsc.edu, pang@cse.ucsc.edu

Abstract

We present three methods of visualizing tensor fields that offer new ways of looking at tensor volumes. All three methods are based on the interaction of simulated light and the tensor field. Conceptually, rays are shot from a certain direction into the tensor volume. These rays are influenced by the surrounding tensor field and bent as they traverse through the volume. The tensor is visualized by both the nature of the bent rays and by the collection of rays deposited on a receiving plate. The former is similar to streamlines, but shows paths of greatest influence by the tensor field. The latter is similar to caustic effects from photon maps, but shows the convergence or divergence of the rays through the tensor volume. We also use the concept of treating the tensor volume as a special lens that distorts an image. Using backward ray tracing through the tensor volume, we generate image distortions that also show internal properties of the tensor field. A key advantage of these techniques is that they can work directly with non-symmetric tensor fields without first decomposing them into components.

Key Words and Phrases: ray casting, caustics, photon mapping, distortion, deformation, refraction, non-symmetric tensor fields

1. INTRODUCTION

Tensor data is more common than you think. It can be found in a number of fields such as: medical imaging [10], fluid flow [5], mechanics [13], and tectonics [17]. Although scientists often have to deal with tensor data sets, the tools available for visualizing them are quite limited. In this paper, we restrict our discussion to second order 3D tensor fields. Visualizing such tensor fields is difficult because each tensor in the field contains nine unique quantities. To incorporate this much information in a single representation is a challenging task. Current methods either show detailed information about the tensor field but only at a few local, discrete points using glyph, or show information about the tensor field in some particular directions.

In this paper, we present three new algorithms for visualizing tensor fields based on interaction of simulated light with the tensor volume. The first is based on ray casting. A set of parallel rays are issued toward the tensor field. As the rays travel through the volume, they are influenced and bent by the tensor field just as light is bent by a gravitational field. In this case, the tensor field is treated as a transforming force, and can be visualized by observing its effects on the rays. That is, the manner in which the rays are bent and deformed reveal the underlying nature of the tensor volume. We refer to this method as *path tracing*. The second method is based on observing the image formed by these bent rays on a receiving plate. Just as caustic effects are observed when light either gets dispersed or concentrated on the bottom of a pool, we can generate similar images of light going through a tensor volume. However, instead of caustics forming due to the undulations on the water surface, caustic effects of tensor fields are produced by how the tensor volume disperses or concentrates the light rays. Thus, these tensor caustic images do reveal the internal convergent or divergent properties of the tensor volume. We refer to this method as *photon distribution* because we are essentially calculating the image based on the distribution of these rays on a receiving plate. The third method assumes that the tensor volume is some kind of lens that distorts an image. It is akin to looking at objects with known shapes using lenses of different shapes and thickness, and then guessing the properties of the lenses. Backward tracing is used to produce the desired effects, and we refer to this method as *lens simulation*. Together, these

three methods offer additional ways of allowing us to analyze the complex nature of tensor fields.

The rest of this paper is organized as follows: Section 2 reviews some previous tensor visualization techniques; Section 3 reviews some tensor transformation techniques to aid with the visualization task; Section 4 presents the three new techniques; Section 5 discusses some implementation issues; Section 6 illustrates the methods and analyzes the results; Section 7 draws conclusions for the new tensor visualization techniques and proposes some future work for tensor visualization.

2. PREVIOUS WORK

There are a few methods for visualizing tensors such as pseudo-coloring, tensor glyphs [3, 9], hyperstreamlines [4], and deformation [2, 12, 15].

A straightforward method is the use of glyphs. The simplest one is the tensor ellipsoid. Here, the tensor is decomposed into three orthogonal eigenvectors, with their corresponding eigenvalues indicating the magnitude along each eigenvector. An ellipsoid is then constructed and oriented according to the 3 eigenvectors and scaled according to the 3 eigenvalues. This works for symmetric tensor fields. More complex glyphs are constructed to show additional features in the tensor fields using flow probes [3]. Another interesting use is in conjunction with other geometric primitives. Hagen et al. [7] proposed the use of generalized focal surfaces to visualize information derived from real, symmetric deformation tensor fields. Directional information is displayed separately by elliptical glyphs placed over characteristic curves. The general drawback of glyphs is their discrete nature. They do not capture the global or continuous nature of tensor fields. And they also require judicious placement to avoid clutter.

Another well known tensor visualization approach is with hyperstreamlines and topology based method [4, 6]. For symmetric tensor fields, the 3 eigenvectors at each point are sorted by their eigenvalues and classified as the major, medium, and minor eigenvectors. For non-symmetric tensor fields, where the 3 eigenvectors are not necessarily orthogonal to each other, the tensor field is first decomposed into a symmetric and an axial component. Hyperstreamlines are then generated by integrating along one of these eigenvector fields, and letting the two other eigenvector and eigenvalue fields modify the shape of a primitive that is swept along the principal hyperstreamline. Because one of the eigenvector field is used for integrating the hyperstreamline, there are two other possible hyperstreamlines that can be generated from each seed location. The understanding of the tensor field is therefore not complete without these two. The drawback of this approach then is that users have to integrate the 3 different views.

Topology based methods use a set carefully selected points, lines and surfaces to divide the tensor field into several small sub-fields, each of which has relatively homogeneous tensors [6]. Trained users can reconstruct the rest of the tensor field by looking at the critical topology. A drawback of this method is that it only uses isotropic tensor as the critical point, which is often less interesting in applications like diffusion tensors.

Direct volume rendering approach applies different transfer function and shading models on the tensor field [8]. The hue-ball and lit-tensor map the tensor anisotropy into view-dependent material. Users can understand the anisotropic tensors by looking at the field from different view points. Because this method is specially designed for diffusion tensor, how it works on other data set is still unknown.

Noise filtering method uses the tensor field to control the filtering process to produce a volumetric scalar field which is then volume rendered [16]. Essentially, the tensor volume is decomposed into a scalar volume by iteratively applying filters that are multiplied with the tensors. Additional post-processing steps are performed to further enhance the directionality of areas with strong eigendirections. The technique works quite well with synthetic data but was less convincing with real data sets.

An important application of tensor visualization is in the medical field. Tensor reconstruction approach rebuilds the neural pathway along the principal eigenvector of the diffusion tensor [1]. In this method, moving least squares regularization successfully overcomes the noise and relatively coarse grid problem in the data set. The tensor tracing is essentially a streamline integration along the principal eigenvector, hence hyperstreamlines. One interesting point is how they chose the locations for seeding the hyperstreamlines.

An alternative approach is to use deformation. Simple objects such as polygons [15] or cubes [12] are advected by a flow field. The deformations on the objects along the advected path show the local stretch, shear, and rigid body rotation at a point. This is generalized in [2] to include idealized objects such as lines, planes, and sub-volumes. It allows users to interactively modify an interrogation vector to see how the tensor field would deform the object. While the user can now see the continuity over the field, the main drawback of this approach is that users can only see the information in one direction at a time. This was further improved in [18] in two ways: (a) allow the normal vectors to change dynamically as the object is being deformed, and (b) allow entire volumes to be deformed by solving a system of linear equations that minimize the energy across a spring system connecting all the cells in the tensor volume. However, one drawback that persisted is that the techniques applies only to symmetric tensor fields. Our goal in this paper is to overcome that constraint. The methods presented in this paper is also in line with our general idea of using deformation to visualize tensor fields. This time we use the idea of deforming simulated light rays, or idealized line segments as suggested in [2].

3. TENSOR BACKGROUND

We review some basic definitions. A tensor field is a field that contains a tensor at each point. We assume that the tensor resides in a structured grid and also assume that it is basically continuous. In a general tensor, each component is a unique quantity. For a 3 dimensional second order tensor field, each location has 9 unique quantities and the tensor at each point is defined as:

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} \quad (1)$$

In many applications, the tensor field is generated from the gradient of a vector field V or the second-order gradient of a scalar field s. In the latter case, the tensor value is symmetric. The tensor fields are represented in the following way:

$$T = \begin{bmatrix} \frac{\partial V_x}{\partial x} & \frac{\partial V_x}{\partial y} & \frac{\partial V_x}{\partial z} \\ \frac{\partial V_y}{\partial x} & \frac{\partial V_y}{\partial y} & \frac{\partial V_y}{\partial z} \\ \frac{\partial V_z}{\partial x} & \frac{\partial V_z}{\partial y} & \frac{\partial V_z}{\partial z} \end{bmatrix} \quad (2)$$

or

$$T = \begin{bmatrix} \frac{\partial^2 s}{\partial x \partial x} & \frac{\partial^2 s}{\partial x \partial y} & \frac{\partial^2 s}{\partial x \partial z} \\ \frac{\partial^2 s}{\partial y \partial x} & \frac{\partial^2 s}{\partial y \partial y} & \frac{\partial^2 s}{\partial y \partial z} \\ \frac{\partial^2 s}{\partial z \partial x} & \frac{\partial^2 s}{\partial z \partial y} & \frac{\partial^2 s}{\partial z \partial z} \end{bmatrix} \quad (3)$$

In the velocity gradient tensor, the local velocity can be expressed as:

$$V = V_0 + \frac{\partial V}{\partial x} \cdot \delta x + \frac{\partial V}{\partial y} \cdot \delta y + \frac{\partial V}{\partial z} \cdot \delta z \quad (4)$$

or

$$V = V_0 + \nabla V \cdot \delta r = V_0 + T \cdot \delta r \quad (5)$$

Given a tensor T and a resolute vector W, W will be transformed into another vector U:

$$U = T \cdot W \quad (6)$$

So a tensor can be considered as a linear transformation from one vector to another vector.

Decomposition

A general strategy for tensor field visualization is to decompose the field into meaningful components. Visualization is then performed based on the components. For example, decomposing a 3 dimensional second order tensor into symmetric and anti-symmetric components results in 6 independent components for the former and 3 independent components for the latter. Other decomposition methods also exist that isolate different properties of the general tensor. The methods presented in this paper can work directly with the general tensors and hence a decomposition step is not necessary.

Iterative method for finding principal eigenvectors

The principal eigenvectors in a general tensor field may be obtained in an iterative fashion [14]. Starting from an initial vector A, the tensor is used to transform it to a different vector. If this process is repeated a few times using:

$$A_{n+1} = \frac{T \cdot A_n}{\|T \cdot A_n\|} \quad (7)$$

until there is very little change in the new vector, then the resulting vector would tend towards the the eigenvector with the largest eigenvalue. This is because the difference between the coefficient of the major eigenvector and those of the others are amplified exponentially during each time step. Although it may start with a relative small value, it becomes dominant after a few iterations. This conclusion is important when we compare our *path tracing* algorithm to hyperstreamlines.

4. APPROACH

We focus on presenting a visualization technique that shows the full information of a tensor field. We do this by treating the tensor field as a transforming force. Objects that enter the tensor space are transformed. In this paper, we introduce simulated light rays into the tensor space. These are bent by the tensor field according to a bending rule described below. These rays represent simulated light rays and may be of different wavelengths, and hence are affected by different amounts.

Bending rule

Initially, a set of rays are cast toward the tensor field. As the rays advance through the volume, their directions are altered by a bending force equal to the product of the ray direction and the local tensor value. The net bending force is used to incrementally bend the direction of the ray as it progresses through the volume. The new direction is normalized at the end of each step.

Assume D_t is the direction of a ray at time step t, P is its position, V is its normalized velocity, $T(P)$ is the tensor

value at P. We use the following formula to update the ray direction.

$$\Delta D_t = T(P) \cdot D_t \tag{8}$$

$$D_{t+1} = \frac{D_t + s(\lambda) \cdot \Delta D_t \Delta t}{\|D_t + s(\lambda) \cdot \Delta D_t \Delta t\|} \tag{9}$$

$$V'(t) \;=\; s(\lambda)(T(P) \cdot V - \frac{V^T \cdot T(P) \cdot V}{V^T \cdot V} V) \tag{10}$$

where $V(t) \;=\; P'(t)$ and $s(\lambda)$ is a scaling parameter as a function of the wavelength, λ, of the ray. It determines the degree to which a tensor field can affect the ray's new direction, and can be expressed in a simple linear form:

$$s(\lambda) = a + b\lambda \tag{11}$$

where a and b are two constants specified by the users. We also refer to this scaling parameter as the *flexibility* of the rays, and is directly related to how easily the rays are influenced by the tensor field. The direction of the ray and the local tensor value are combined by taking their product. A fraction of this product is added to the ray direction to get the new direction vector. The new direction is then normalized to produce the true direction. From Equation 9, we see that this transformation rule gradually drags the ray direction towards the major eigenvector after each time step. This property is useful when interpreting the result. Equation 10 expresses the same idea in differential form.

Compare the bending rule to the iterative method for eigenvectors, we know that when $s(\lambda) \to \infty$, the normalized version of $P'(t)$ is the principle eigenvector of $T(P(t))$. So the major hyperstreamline is a special case of the *path tracing* algorithm when $s(\lambda)$ is very large. As $s(\lambda)$ decreases, we see influences from the weaker tensor directions. In general, *path tracing* gives users the flexibility to choose between the tensor eigenvector streamlines and integration and comparison of tensors from different components of the field.

The path of a ray is sensitive to the property of the bending rule. In particular, the scaling parameter, $s(\lambda)$, in our algorithm. A slight change in the λ may cause noticeable variations in the result. The advantage of this property is that we can explore different parts of the data using this ray tracing scheme. However, the drawback is also obvious: the results need to be interpreted in conjunction with the choice of λ.

Path Tracing

The bending rule defined on the tensors changes the ray direction in each time step. We can understand the nature of the underlying tensor field by looking at these bent rays from different perspectives. But for a particular set of incident parallel rays, how to present these rays is still a hard task. The most straightforward method is to draw all the bent rays within the tensor field simultaneously. This is similar to the streamline algorithms in vector visualization. To encode the information in the other eigendirections along the ray trajectory, we use elliptical cross sections. These ellipses can be rendered individually or assembled to form a tube for visualization.

Within a symmetric tensor field, where the eigenvectors are all perpendicular to each other, the path tracing algorithm can reproduce hyperstreamlines. Using a highly flexible rays, the current ray direction will align with the major eigenvector within a short distance from the seed point. Likewise, the plane of the cross sectional ellipses will be perpendicular to this direction as well. And its shape is determined by their eigenvalues. This is exactly the same as hyperstreamlines. With this method, we can see how the tensor field bends and affects the rays' paths and cross sectional shapes.

While the path tracing algorithm can reproduce hyperstreamlines, it is different from that algorithm as described by Hesselink [4]. First, we do not need the tensors to be symmetric. The bending rules can be applied directly on any tensor fields to show the shearing, converging and rotating properties. Secondly, even in symmetric tensor fields, the algorithm gives users more flexibility to control the path traces. When the rays are getting closer, we can claim the tensors have higher convergent components, which is rarely shown by the hyperstreamline algorithm. The path tracing algorithm clearly shows the convergent and divergent properties of the tensor field. When the tensors are convergent in a local area, we find the path traces are also convergent.

Unlike the streamline algorithm in vector visualization, the bent rays in the path tracing visualization may cross each other. That is, there is no inherent guarantee in the formulation that the rays cannot intersect itself or one another. As the number of rays increases, the visual clutter also increases. Thus, some experimentation and interaction is necessary to find an acceptable balance in analyzing a particular data set. We also provide alternative methods that does not have the problem of visual clutter to complement this technique.

Photon Distribution

Using the same bending rule, we define another rendering algorithm, *photon distribution*, to visualize the effect of the tensor field upon the rays. Aside from the clutter problem, we also note that in the *path tracing* algorithm, the paths are different when we change the ray flexibility. This *photon distribution* algorithm takes advantages of the continuity in the ray paths and combine the photons from different wavelengths together.

This algorithm is inspired by the mechanism of a prism.

A prism can produce colorful patterns from a beam of white light. White light is composed of lights of different wavelengths. Although incoming rays may have the same incident angle, light of different wavelengths have different paths within the prism. When they exit out of the prism and hit on the receiving plane, they generate a colorful pattern.

The photon distribution algorithm is analogous to the mechanism of light being separated and bent as it goes through a prism. We position the tensor volume, which acts as a "prism", between a receiving plane and where the rays are shot from. The receiving plane is used to collect information about where the rays or photons land after traversing through the tensor volume. We then shoot beams of white light, each of which is represented as having equal amounts of red, green and blue components, to the tensor field. Since lights of different wavelengths have different indices of refraction, their bent paths will be slightly different. When the rays leave the tensor field, the three color components that came from the same direction and issuing position end up with different positions on the receiving plane. These intersections are deposits of photons in much the same fashion as photon mapping and form a distribution pattern on the receiving plane. By combining these distribution patterns, we get an image that represents the nature of the underlying tensor field, especially the converging and diverging properties.

Lens Simulation

Another method to visualize the bent rays is analogous to the mechanism of lenses. We assume that the tensor field is some kind of lens and we are looking at an image through this lens. The image that we see will depend on the shape and internal properties of the lens, and will generally produce some distortions into our image. By looking at the distorted images from different perspectives, we gain an understanding of the properties of the lens, and hence the underlying tensor volume.

To simulate this process, a set of parallel rays is cast from the image toward the viewer and through the tensor fields. This is analogous to the backward ray tracing algorithm. While the rays are within the tensor field, their directions are altered by the bending force defined on the tensors. They leave the field in different locations and orientations. These rays eventually hit a receiving plane, from where we generate a distorted image. The pixels on the distorted image are colored according to the color of the corresponding pixels from the original image. In our implementation, the texture coordinate of each point is assigned to the original pixel, and neighborhood information is preserved.

In lens simulation, although we have no prior knowledge of the inner structure and surface shape of the lens, we can understand its nature by looking at the distorted picture. The way that the picture is transformed under the effect of the lens reveals the nature of the underlying lens. An enlarged area of the picture corresponds to a convex part of the surface or a high mass area; a reduced or compressed area corresponds to a concave part or a low mass area. By looking at the picture through the lens from different angles, we can get a better idea of both the inner mass distribution and the surface shape of the lens.

The lens simulation algorithm does not have the problem of visual clutter. However, one must be careful when interpreting the picture because each ray is the integrated sum of the effects of the tensors along the path of the ray. We need to use several images from different angles to determine the tensor properties of a local area.

The advantage of this algorithm is that it shows a spatially continuous depiction of the properties of the tensor field. Unlike the path tracing algorithm, which use 1D line glyph to represent the tensors, the lens simulation uses an image to reveal the 2D features of the same field. Users can observe the continuous changes through the image deformation.

The three algorithms just described can explore different parts of the data at the same time. They use the same bending rule on the rays. The paths of the rays are the integral of the bending forces along each path. So it is sensitive to the parameters used in the bending rule. All of the three rendering algorithms, *path tracing*, *photon distribution* and *lens simulation* are able to show the converging and diverging properties of the tensor field.

5. IMPLEMENTATION ISSUES

In this section, we discuss some implementation details related to the three algorithms in the paper

Preprocessing

One of the most important properties of *ray casting* is that is is very close to the major *hyperstreamlines* when the ray flexibility, $s(\lambda)$ is very large. We get this conclusion from the bending rule and the iterative method for eigenvectors. Because of lack of sign for eigenvectors, the integral of *ray casting* often keeps changing sign when the eigenvalues is negative. So we always keep the sign of the eigenvalues as positive. Assume a tensor is T, and its eigenvectors and eigenvalues are $e_i, i = 1.2, 3$ and $\lambda_i, i = 1, 2, 3$. The tensor, T_p, after preprocessing is:

$$T_p = E \cdot \begin{pmatrix} |\lambda_1| & 0 & 0 \\ 0 & |\lambda_2| & 0 \\ 0 & 0 & |\lambda_3| \end{pmatrix} \cdot E^{-1}$$
$$E = \begin{pmatrix} e_{1x} & e_{2x} & e_{3x} \\ e_{1y} & e_{2y} & e_{3y} \\ e_{1z} & e_{2z} & e_{3z} \end{pmatrix} \quad (12)$$

Elliptical cross section

In the *path tracing* algorithm, the trajectory of the path

encodes the major eigendirection. The tensor information of the other two directions can be encoded in the shape of the cross section of the path. For this, the ellipse is used. Given a tensor T, a ray traveling in the direction N and a unit sphere S. The ellipsoidal tensor glyph G is the unit sphere S transformed by the tensor T. We are trying to get the cross section of G with the normal vector of N. Assume this cross section is Y, which is transformed from X on the unit sphere by T. Then we have,

$$0 = N \cdot Y = N \cdot (TX) = (N^T T)X = (T^T N) \cdot X \qquad (13)$$

It is easy to get the cross section of the unit sphere, X, with the normal vector of $T^T N$. In our algorithm, we sample from the point set of X, then transform them by T to Y, to produce the cross section of the tensor ellipsoid glyph, G.

Stability

In the updating rule, we set the step length small enough not letting anything big happen during one time step. The step length and ray flexibility satisfy the following inequality:

$$s(\lambda)\Delta t < \frac{1}{2}\min_P \frac{G(P)}{|e(P)|} \qquad (14)$$

where P is a point, $G(P)$ is the length of any of its neighboring grid lines, $e(P)$ is the magnitude of any of its eigenvalues. From equation (14), we know if we set $s(\lambda)$ as a very large number, the time step must be very small to make the integral stable. Using implicit method could address this problem when large ray flexibility for complicated data set is necessary.

Photon distribution density

In the *photon distribution* algorithm, one can get a more accurate picture of the photon distributions by shooting more rays. However, this is rather expensive and not always practical. Instead, we treat all the photons as samples from a probability distribution. Different density estimators are employed to get approximate the photon density of a local area. The color magnitude is then mapped to the density of the photons in the corresponding wavelength in a certain area. The more rays land on a local area of the receiving plane, the higher its density.

The density estimator is based on Gaussian distribution. With Gaussian estimators, each photon in the sample set has an effect on its local neighborhood. Its effect decreases as the distance increases. Assume we have a point P, and its magnitude is $M(P)$, R is the set of points on the receiving plane.

$$M(P) = \sum_{q \in R} \frac{e^{-\|P-q\|^2}}{2\tau^2} \qquad (15)$$

where τ is responsible for the radius of influence.

Although this formula is defined on a global scope, the Gaussian function has a conceivable effect only in an area where the distance from P is less than 3τ. So, we only do the magnitude computation in a local area around the points hit by the rays on the receiving plane. This algorithm is fast and stable. Its drawback is that it is undefined in some areas with very sparse ray hits.

Color mapping

The color of the path traces can be mapped to a number of parameters. In our experiments, we map the color of the path traces the magnitude of the bending force as follows:

$$Color(P) = \|T(P) \cdot D_t\| \qquad (16)$$

6. RESULTS

We tested the three algorithms on two data sets. The first one is the well known single point load Boussinesq data set. This is a standard stress tensor data set on a regular grid. It is simple and thoroughly studied, so it can be used to verify the correctness and usefulness of our algorithms. The second data set is a velocity gradient tensor field derived from the flow past a cylinder with a hemispherical cap. This is the same data set used in earlier papers on tensor visualization, e.g. [11], and is a good benchmark for comparison as well.

Path tracing results

Figure 1 shows the basic results from the *path tracing* algorithm. In both, the rays are coming from the same direction as the point load, but the results are viewed from two different view points. Figure 1(a) is a side view of the tensor field and Figure 1(b) is taken from a "worm's" eye view. Combining these two pictures, we can observe the 3D structure of the path trajectories. We note that the path traces move away from the central axis, which means the tensors are divergent in the point load direction.

Figure 2 shows two images this time from the same view points, but with rays entering the tensor volume from different directions. The view point of both pictures are taken from the side of the tensor field. In Figure 2(a), the rays are cast opposite the point load direction. In Figure 2(b), the rays are coming from the left side of the image. Note that on the left image, the trajectories are convergent with respect to the casting direction. That is, the rays are converging towards the point load direction. One the right, the effects of the tensor field are not as prominent because the flexibility is relatively

low. Hence, the rays end to stay in the direction that they were cast.

Figure 3 shows four images from the hemisphere data set. The rays integrate all the high shear components of the tensors along their paths, and thereby reveal the detachment of the flow above the hemisphere. The paths can be rendered in a number of ways: shaded tubes, wireframe tubes, and disconnected cross sectional ellipses. The wireframe tubes seem to be effective in showing details of the path and cross sectional shapes in a medium density field of rays. We see the rays get over the hemispherical cap and converge at the other end. It shows that the compressing force also gets around the geometry at the same time. The sudden change in color and shape on the tubes also marks out the change of homogeneous sub-region.

Figure 4 shows two images from the hemisphere data set but with much larger ray flexibility. Figure 4(a) is taken from the side view, while Figure 4(b) is taken from the top view. Rays are cast into the field from the right side of the images. We can observe that these lines are very close to the major hyperstreamlines found in Hesselink's work.

Photon distribution results

Figure 5 illustrates the result from the *photon distribution* algorithm. In Figure 5(a), an array of photons of different wavelengths are cast toward the tensor field. Their locations on the receiving plane are treated as samples from a spatial probability distribution. We then use a 2D Gaussian density estimator to map the photon density to color magnitude. The ray flexibility for red, green and blue color are $(0.1, 0.2, 0.3)$ respectively. The blue color is most affected by the tensor fields, while the red color is relatively "directly" projected to the receiving plane.

From the hole in the center area of Figure 5(a), we can easily identify the divergence of the tensors along the point load direction. Figure 5(b) shows the view from the bottom of the point load data. Note the brighter area in the middle which corresponds to a higher density of rays. This is because the photons along the rays are deflected towards the central axis, and at a greater magnitude for rays that are closer to the central axis. These two images correspond directly to the path tracing images Figure 1(a) and Figure 2(a) respectively. Figure 5(c) illustrates the result from the RGB ray tracing synthesis. Recall that the red component has a large scaling factor, green has a medium factor and the blue light has the smallest scaling factor. The patterns from the three components are then combined into one picture. We note that red light has a wider area. This is consistent with the divergence of this tensor field.

Figure 6 illustrates the same *photon distribution* algorithm for the hemisphere data set. Four photon images are taken from different perspective to get a full view of the entire tensor field. Figure 6(a) is imaged from a front view. We see a white spot on top of the cylinder, which corresponds to the critical point observed in topology based tensor visualization methods. The blue color in the outer layer reveals the divergence in that area. Figure 6(b) is imaged from the top view. Despite the shock wave pattern around the hemisphere, the white spots also correspond to a highly converging area. Figure 6(c) is imaged from the side view; The overall color in this image is quite dark. But we still see several bright areas. These areas reveal where the tensor field is forcing the photons to converge. Figure 6(d) is imaged from an oblique view. In this image, the white bar below the cylinder is the same bright bar in the side image.

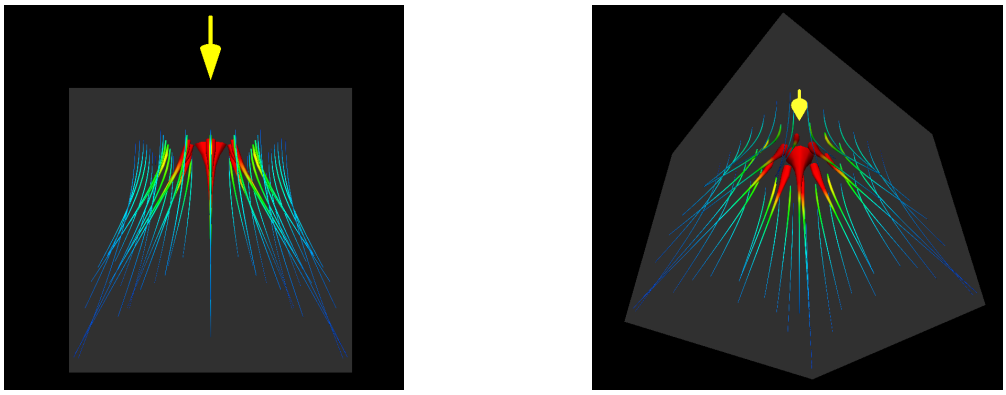
Lens simulation results

Figure 7 illustrates the results from the *lens simulation* algorithm on the point load data. Figure 7(a) is the original undistorted image. We use a regular checkerboard pattern so that it is easy to identify correspondence in the distorted image. Figure 7(b) is taken from the point load view and Figure 7(c) is taken from the direction opposite to the point load. They clearly show the divergence and convergence of the tensors along the central axis from opposing view points. Because the *lens simulation* works as an inverse method to the *photon distribution* algorithm, the results also look reversed. Figure 7(d) is taken from the side of the tensor field. We note that most areas in this image are not twisted, and do not self intersect.

Figure 8 illustrates the results from the *lens simulation* algorithm applied to the hemisphere data set. Again the original image is a regular checkerboard pattern, while the image through the tensor volume lens is registered as a distorted image on the receiving plane. The relationship of the receiving plane with respect to the placement of the original image is illustrated in Figure 8(a). Note that the placement of the original image acts as a cutting plane that can be used to slice through the volume and control the portion of the tensor volume that can affect the distorted image. Figure 8(b) is taken from the top view where a bit of the compressive shock wave is visible. Figures 8(c) and 8(d) are taken from the side. In Figure 8(c), the original texture is placed abutting against the geometry of the cylinder so that the rays are not obstructed by the geometry. In figure 8(d), the original texture is placed halfway through the geometry. The vertical profile of the compressive shock is visible in these two images.

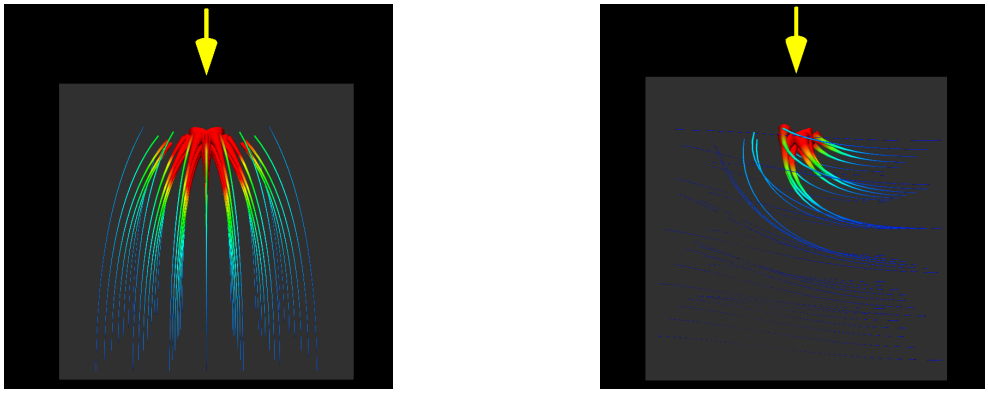
7. CONCLUSIONS

Several methods based on path tracing for visualizing tensors are presented in this paper. *Path tracing* algorithm is good at showing the detailed direction information of the tensors. But it suffers from the same disadvantage as the other 3D streamline algorithms in that it produces heavy visual clutters. Although we employ a specially designed illumination model to address this problem, it is still difficult



(a) side view (b) worm's eye view

Figure 1: *Path tracing algorithm on the single point load dataset. The rays are coming from the point load direction and are diverging from there.*



(a) rays from bottom (b) rays from side

Figure 2: *Path tracing algorithm on the point load dataset. Rays are cast from bottom in the left image, and from the left side in the right image. On the left, the rays are converging towards the point load. On the right, the ray flexibility is low and hence is not affected as much by the tensor field.*

for users to understand the tensor field when the number of rays is high. The *lens simulation* and *photon distribution* algorithms present the visualization results in a way similar to the lens or the prism. We can identify the direction information of the tensors by observing several images taken from different view points around the tensor field. In the immediate future, we will evaluate these methods on more complex data sets.

ACKNOWLEDGEMENTS

We would like to thank members of the Advanced Visualization and Interactive Systems laboratory at UCSC for some utility code to get the project started. This work is supported in part by NSF ACI-9908881 and NASA Cooperative Agreement NCC2-1260.

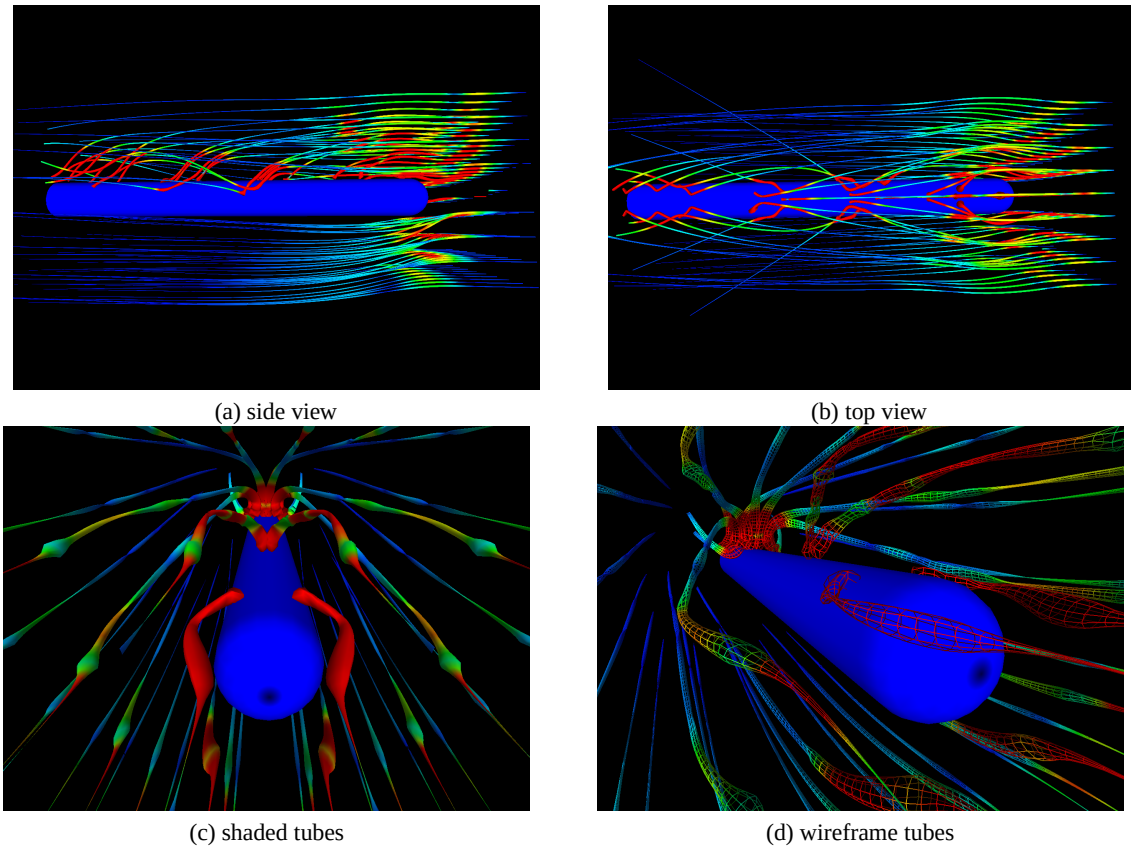
References

1. Leonid Zhukov an Alan Barr. Oriented tensor reconstruction: Tracing neural pathways from diffusion tensor MRI. In *Proceedings of Visualization 02*, pages 387–394, Boston, 2002.

2. Ed Boring and Alex Pang. Directional flow visualization of vector fields. In R.D. Bergeron and A.E. Kaufman, editors, *Proceedings of Visualization 96*, pages 389–392. ACM, October 1996.

3. W.C. de Leeuw and J.J. van Wijk. A probe for local flow field visualization. In G.M Nielson and D. Bergeron, editors, *Proceedings IEEE Visualization '93*, pages 39–45. IEEE Computer Society Press, 1993.

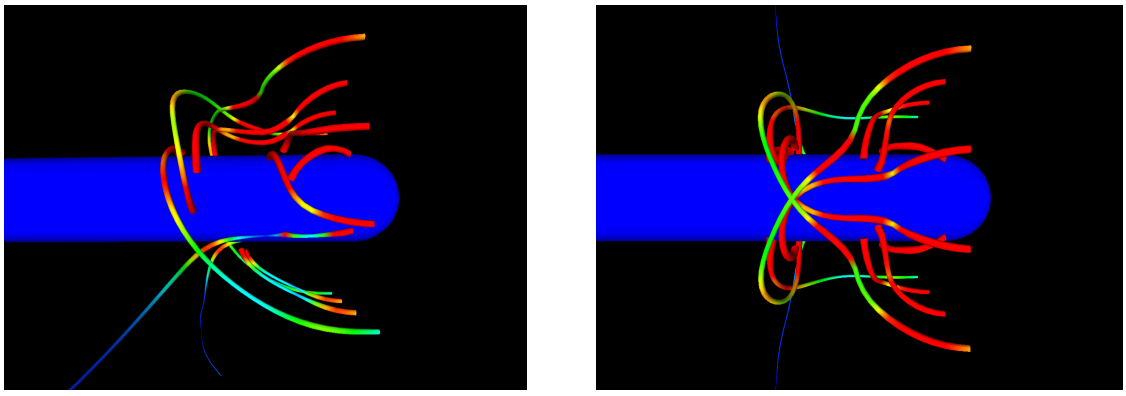
4. T. Delmarcelle and L. Hesselink. Visualization of second order tensor fields and matrix data. In *Proceedings*



(a) side view (b) top view

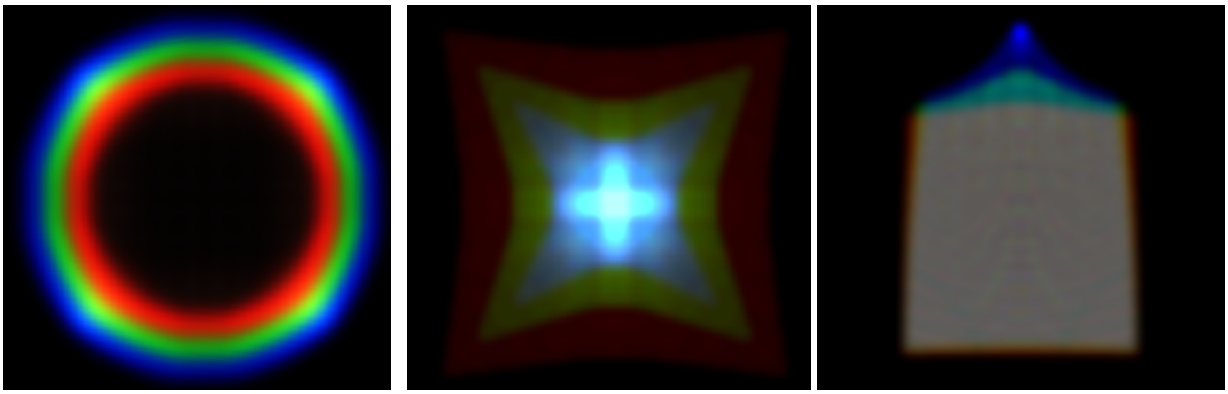
(c) shaded tubes (d) wireframe tubes

Figure 3: *Path tracing algorithm on the hemisphere dataset. Rays are coming going towards the hemispherical cap. Most traces converge at the rear end of the cylinder. It reveals the compressing directions in this dataset. Note how the tubes gets thicker when they approach the hemisphere cap and then get thinner as they reach the top of the cylinder. This shows the compressing forces are quite larger in the vertical direction in the cap area than in the cylinder area. We also note that in the area where the tubes are converging near the rear end, the tubes are almost flat, which illustrates higher anisotropy.*



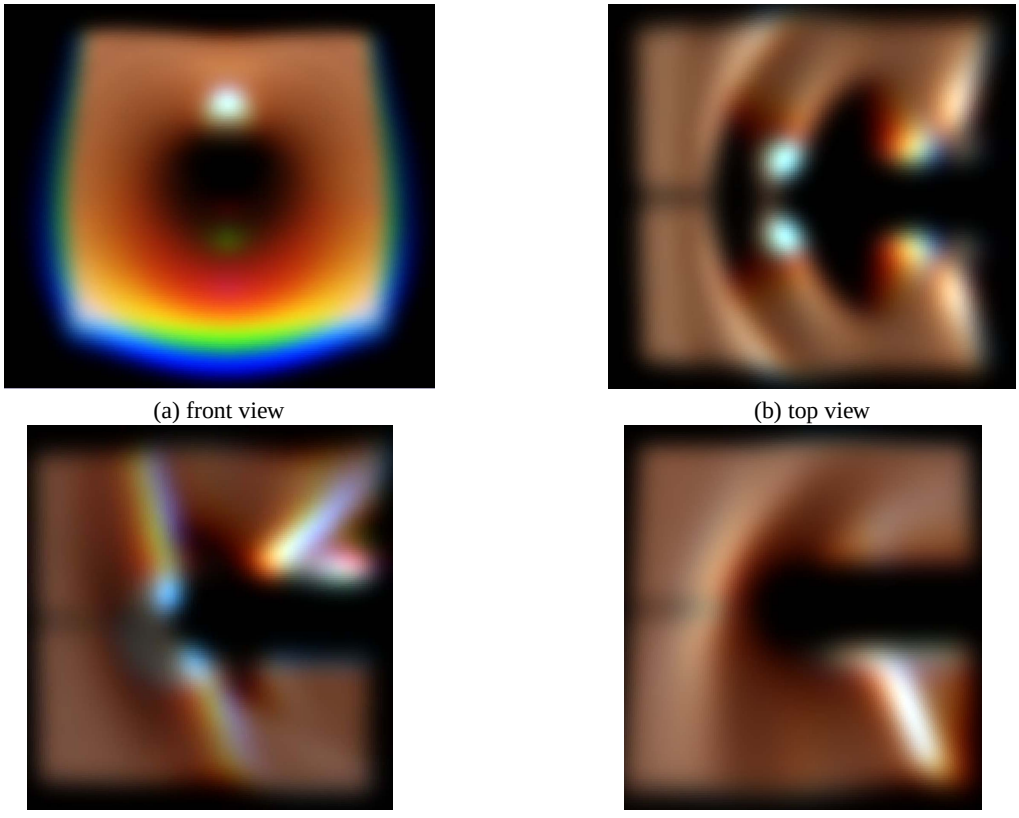
(a) side view (b) top view

Figure 4: *Path tracing algorithm on hemisphere dataset using large ray flexibility. Different view points are used to present the deformed rays. The pattern of the rays is very close to Hesselink's result on the same viscous stress tensor data set. They both get across the hemispherical cap and converge at the other end in a similar way.*



(a) top view (b) bottom view (c) side view

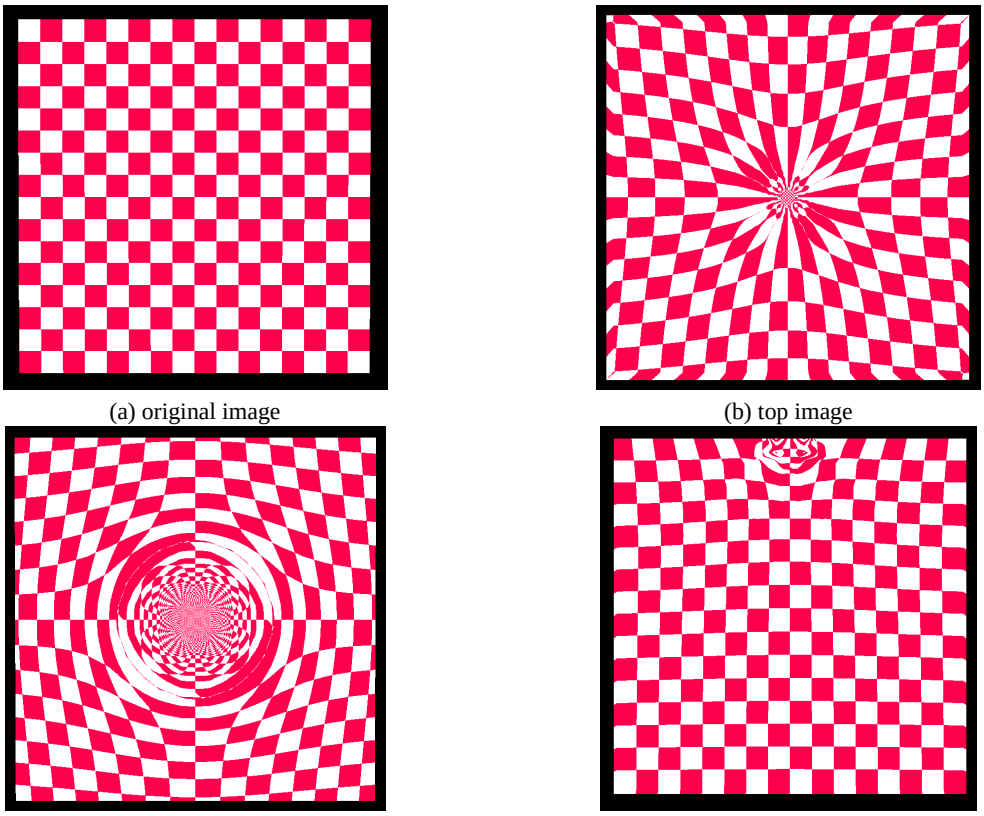
Figure 5: *Photon mapping algorithm on point load data.*



(a) front view (b) top view

(c) side view (d) oblique view

Figure 6: *Photon mapping algorithm on hemisphere data.*



(a) original image (b) top image

(c) bottom image (d) side image

Figure 7: *Lens simulation algorithm on point load data.*

of Visualization 92, pages 316–323, CP–34, 1992.

5. T. Delmarcelle and L. Hesselink. Visualizing second-order tensor fields with hyperstreamlines. *IEEE Computer Graphics and Applications*, 13(4):25–33, July 1993.

6. T. Delmarcelle and L. Hesselink. The topology of symmetric, second-order tensor fields. In *IEEE Visualization*, pages 140 –147, 1994.

7. H. Hagen, S. Hahmann, and H. Weimer. Visualization of deformation tensor fields. In G. Nielson, H. Hagen, and H. Muller, editors, *Scientific Visualization: Overviews, Methodologies, Techniques*, pages 357–371. IEEE Computer Society, 1997.

8. G. Kindlmann, D. Weinstein, and D. Hart. Strategies for direct volume rendering of diffusion tensor fields. *Visualization and Computer Graphics*, 6(2), 2000.

9. R.M. Kirby, H. Marmanis, and D.H. Laidlaw. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of Visualization 99*, pages 333–340, San Francisco, 1999.

10. David Laidlaw, Eric Ahrens, David Kremers, Matthew Avalos, Russell Jacobs, and Carol Readhead. Visualizing diffusion tensor images of the mouse spinal cord. In *Proceedings of Visualization 98*, pages 127–134, 1998.

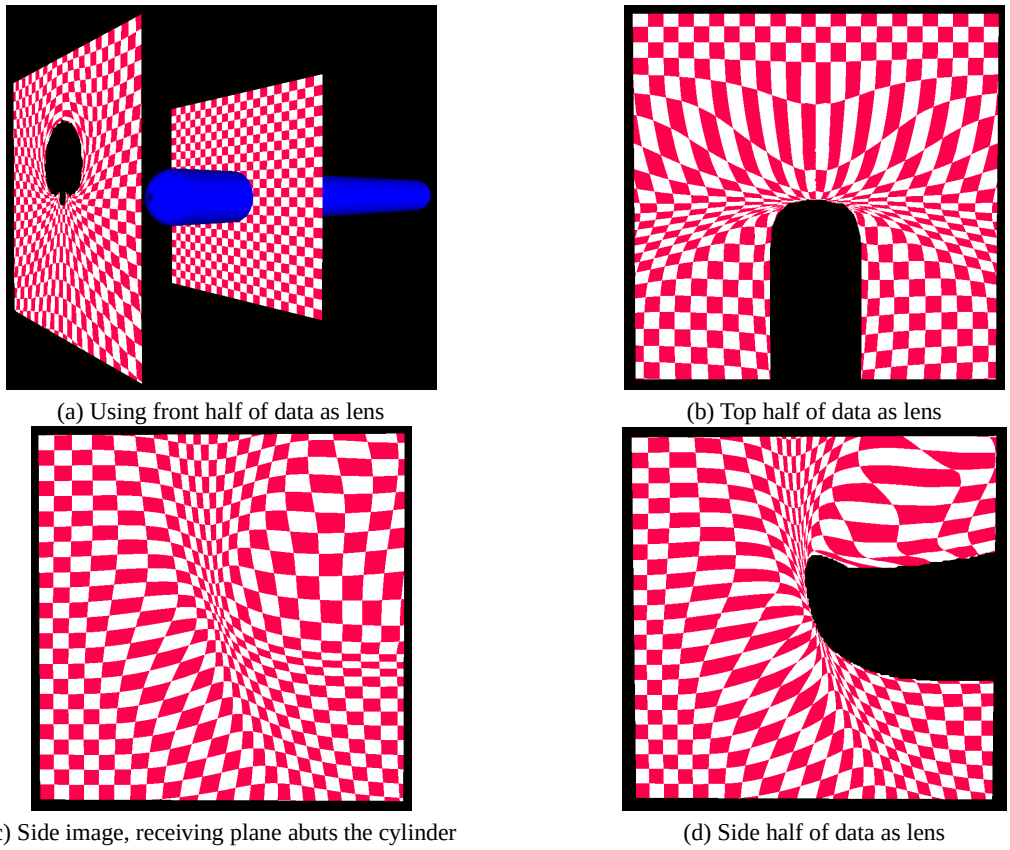
11. Y. Lavin, Y. Levy, and L. Hesselink. Singularities in nonuniform tensor fields. In *Proceedings of Visualization 97*, pages 59–66, 1997.

12. Xundong Liang, Bin Li, and Shenquan Liu. The deformed cube: a visualization technique for 3D velocity vector field. In *Image Analysis Applications and Computer Graphics. Third International Computer Science Conference. ICSC'95*, pages 51–58. Springer, 1995.

13. L. Malvern. *Introduction to the Mechanics of a Continuous Medium*. Prentice Hall, 1997.

14. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.

15. W. Schroeder, C. Volpe, and W. Lorensen. The stream polygon: A technique for 3D vector field visualiza-



(a) Using front half of data as lens

(b) Top half of data as lens

(c) Side image, receiving plane abuts the cylinder

(d) Side half of data as lens

Figure 8: *Lens simulation algorithm on hemisphere data.*

tion. In *Proceedings: Visualization '91*, pages 126–132. IEEE Computer Society, 1991.

16. Andreas Sigfridsson, Tino Ebbers, Einar Heiberg, and Lars Wigstrom. Tensor field visualization using adaptive filtering of noise fields combined with glyph rendering. In *Proceedings of Visualization 02*, pages 371–378, Boston, 2002.

17. D. L. Turcotte and G. Schubert. *Geodynamics*. Cambridge University Press, 2001.

18. Xiaoqiang Zheng and Alex Pang. Volume deformation for tensor visualization. In *Proceedings of Visualization 02*, pages 379–386, Boston, 2002.