HyperLIC

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 introduce a new method for visualizing symmetric tensor fields. The technique produces images and animations reminiscent of line integral convolution (LIC). The technique is also slightly related to hyperstreamlines in that it is used to visualize tensor fields. However, the similarity ends there. HyperLIC uses a multi-pass approach to show the anisotropic properties in a 2D or 3D tensor field. We demonstrate this technique using data sets from computational fluid dynamics as well as diffusion-tensor MRI.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques;

Keywords: hyperstreamlines, LIC, symmetric tensors, anisotropy, animation, direct volume rendering

1 INTRODUCTION

Tensor data is useful in many medical, mechanical and physical applications. Second order tensors in 3D is a 3x3 matrix that contains nine unique quantities, or six for the case of real symmetric tensors. To help comprehend such a large volume of information remains a difficult challenge for visualization research.

In this paper, we introduce a new method for visualizing symmetric tensor fields. It works on 2D tensor fields in 2D manifolds or 3D tensor fields in 3D manifolds. 3D tensor data in 2D manifolds have to be projected to the manifolds using the tensor transformation and projection introduced later in this paper. The visualizations are specially good at showing the anisotropy in the tensor fields and the resulting images resemble those of LIC. Similar to LIC, a white noise texture is needed. Conceptually, every pixel in the resulting visualization is calculated using an area (or volume, for 3D) averaged noise texture. The shape of the area (or volume) is determined by the local tensor field. Carrying this process out on a point by point basis will result in a blurred image as this is akin to low pass filtering. A better strategy is to think of the process as placing primitives such as squares or circles in 2D, cubes or spheres in 3D, along the path of a hyperstreamline. These primitives are deformed by the tensor field, and the resulting swept area or volume identifies the parts of the noise texture that will contribute to the intensity of a pixel or voxel.

What we are seeing in the HyperLIC visualizations is the anisotropy in the tensor field. Anisotropy in 3D tensor fields can be classified into three types: (a) linear or highly anisotropic characterized by the dominance of one eigenvalue, (b) planar characterized by two roughly equal eigenvalues, and (c) spherical or isotropic characterized by three roughly equal eigenvalues. HyperLIC is particularly good at distinguishing between linear and spherical tensor regions because of the strong contrast between the sharp and smooth features respectively. Planar tensor regions do not stand out as dramatically compared to these two when viewed from different directions.

Digital images and animations can be accessed online at: www.cse.ucsc.edu/research/avis/hyperlic.html.

2 RELATED WORK

We review several classic and recent works that are related to this paper in one way or another. These are: hyperstreamlines [Delmarcelle and Hesselink 1993], adaptive filtering of noise fields [Sigfridsson et al. 2002], oriented tensor reconstruction [Zhukov and Barr 2002], direct volume rendering of diffusion tensors [Kindlmann et al. 2000], and line integral convolution [Cabral and Leedom 1993].

Hyperstreamlines were introduced by Delmarcelle and Hesselink in 1993. A tensor field is first decomposed into three eigenvector fields. Hyperstreamlines are essentially streamlines constructed from one of the eigenvector fields. The other two eigenvector fields are then encoded as changes in the cross section along the streamline. From any seed point, three hyperstreamlines can be generated using one of the three eigenvector fields for the streamlines and the other two for the cross section. In this sense, the technique does not provide a global view and users need to mentally fill in what is happening with the tensor field even in the vicinity of the seed point. Like streamlines, one cannot seed too many hyperstreamlines as clutter becomes an issue. For non-symmetric tensors, the rotational components are encoded as “wings” along the main hyperstreamlines. HyperLIC is similar to how hyperstreamlines handle the symmetric portion of the tensor field in the following manner – the volume swept out by a hyperstreamline roughly corresponds to the volume of the noise texture used by HyperLIC to calculate the intensity value at the seed point.

This brings us to the LIC algorithm for visualizing vector fields, introduced by Cabral and Leedom in 1993. Given an input vector field and a noise texture with the same dimensions, a LIC image is generated by calculating a streamline at each point, and then calculating a weighted average of the noise textures along the streamline, to produce the intensity value at the seed point. Interrante has extended LIC to 3D and presented ways to visualize flow within the 3D volume [Interrante and Grosch 1997]. HyperLIC is similar to LIC in that it calculates a weighted average of noise texture values along a streamline. However, instead of simply using noise texture values along the streamline, we use noise texture values in the vicinity of the streamline as well. This local vicinity is defined by how tensors along the streamline deform the space around it. Specifically, unlike hyperstreamlines which uses only one of the eigenvector fields at a time to integrate streamlines, HyperLIC uses both (2D) or all three (3D) eigenvector fields to define the local deformation space.

More recently, [Sigfridsson et al. 2002] introduced an algorithm for visualizing symmetric tensor fields by iteratively applying an adaptive filter to directionally smear a noise texture in the frequency domain. Several discrete predefined directional filters are employed to categorize the continuous tensor anisotropy. However, the discretization of the filter orientation may be a potential drawback. If an anisotropic linear tensor falls in between a pair of oriented filters, it can only be described by their joint effects. Because of this, some contrast is lost in the result.

Recently, attention has focused on how to visualize diffusion tensor MRI images. One of the challenges is how to deal with the noisy nature of the tensor field particularly when tracing neural pathways [Zhukov and Barr 2002]. Tracing essentially involves integrating a streamline using the principal eigenvector. In that method, moving least square regularization successfully overcame the noise problem and the relatively coarse grid used in the data set. Similar to hyperstreamlines, seeding is an issue that needs to be addressed in a more general fashion. Since the neural pathway tracing was to highlight regions of high anisotropy, seeds were naturally initiated where the linear tensors were high. Like hyperstreamlines, the method provides crisp visualization of streamlines along an eigenvector, but does not provide a continuous, global view of the tensor field.

One approach that does attempt to provide a continuous global view is the adaptation of direct volume rendering to tensor fields presented by [Kindlmann et al. 2000]. Like [Zhukov and Barr 2002], the tensor field is first analyzed with respect to its anisotropy and classified into three continuous categories: linear (anisotropic), planar, and spherical (isotropic) tensors. This property of the tensor field is then used as barycentric coordinates of a triangular transfer function that highlights regions of different anisotropic properties. Further enhancements are then provided using lit-tensors mixed with opacity gradient shading and hue-balls combined with deflection mapping [Kindlmann and Weinstein 1999]. The deflection mapping strategy seems to provide the most dramatic results, but would also depend on the granularity of the textures used.

3 METHODS

In this section, we present the basic idea behind the HyperLIC algorithm, and show how it is refined to provide a more continuous representation of the anisotropy in the tensor field. This is followed by a description of how HyperLIC is actually implemented as a more efficient multi-pass approach. The descriptions are first presented for 2D tensor fields. Subsequent discussions present how the multipass algorithm can be easily extended to 3D. Other issues such as how to deal with sign indeterminacy, animation, and rendering of 3D HyperLIC results are presented in this section as well.

Before we describe HyperLIC in detail, we introduce some preparatory transformations on the data.

3.1 Tensor Processing

Some preprocessing may be necessary to prepare the data for the HyperLIC algorithm. Examples include: if the data is defined in a curvilinear grid or if there is distinction between computational and physical coordinate systems; a 2D manifold is to be extracted from a 3D tensor field; or a sharper contrast is desired from the HyperLIC algorithm.

3.1.1 Tensors in Curvilinear Grids

It is quite common for computational fluid dynamics applications to distinguish between computational space and physical space. For example, if we want to study the tensors on a wing geometry, this would correspond to a slice in computational space. Alternatively, one may want to study the tensors on a slice through physical space. In either case, 3D tensors need to be projected onto a surface. This surface can be expressed as $S(u,v)$, where u and v are computational coordinates. We also need to find a transformation between tensors in physical and computational spaces. Let the three principal axes be: $Q_1 = \frac{\partial S}{\partial u}$, $Q_2 = \frac{\partial S}{\partial v}$ and $Q_3 = Q1 \times Q2$. Also define:

$$W = \begin{pmatrix} Q_{1x} & Q_{2x} & Q_{3x} \\ Q_{1y} & Q_{2y} & Q_{3y} \\ Q_{1z} & Q_{2z} & Q_{3z} \end{pmatrix} \tag{1}$$

Assume the tensor is represented as P in physical space and represented as C in computational space. We want both P and C to transform a unit sphere in their own space, S_p and S_c respectively, into the same ellipsoid in physical space. Because S_p and S_c are both unit spheres, $S_p = R \cdot S_c$, where R is a unitary matrix such that $R^T \cdot R = I$. This can be expressed as:

$$P \cdot S_p \quad = \quad P \cdot R \cdot S_c \tag{2}$$
$$P \cdot S_p \quad = \quad W \cdot C \cdot S_c \tag{3}$$

where we obtain

$$P \cdot R \;=\; W \cdot C \tag{4}$$

and solving for C

$$C \;=\; W^{-1} \cdot P \cdot R \;=\; POLAR_S(W^{-1} \cdot P) \tag{5}$$

where $POLAR_S(X)$ is the symmetric part of X in a polar decomposition. This transformation guarantees that C in computational space has the same effect as P in physical space.

The justification for the transformation above is because a matrix M can always be represented as the product of a symmetric matrix T_s and a unitary matrix R.

$$M = T_s \cdot R \tag{6}$$

And if the transformation matrix W between the physical space and the computational space is unitary, the transformed tensor C from Equation 5 is the same as that from a classical tensor transformation. When W is not unitary, a tensor resulting from a classic transformation may be asymmetric, while the tensor from Equation 5 is always symmetric.

3.1.2 Tensor Projection

Once the tensor is in the computational space, we need to project it onto the surface, $S(u,v)$. The first two axes, Q_1 and Q_2 are on the surface while Q_3 is perpendicular to this surface. So we discard all the components associated with Q_3, which results in:

$$C_2 = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \tag{7}$$

where C_2 is now a 2×2 tensor. Through this projection, a planar tensor along the surface is expressed as a 2D tensor without loss of information.

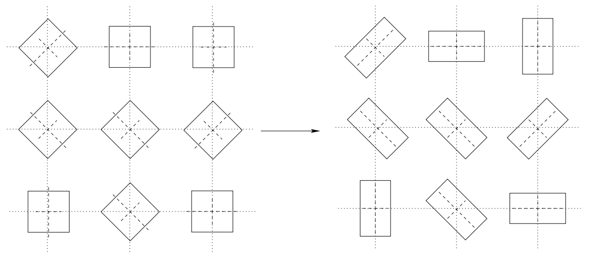


Figure 1: Squares primitives aligned with eigenvectors.

3.1.3 Tensor Normalization and Rescaling

Tensor normalization and rescaling allow us to vary the contrast level of the resulting image. Before normalization, we define the largest absolute eigenvalue of a tensor as its magnitude which we denote by l. The eigenvector associated with the largest absolute eigenvalues is defined as the principal axis of a tensor. For all the tensors, we multiply them with the inverse of their magnitude, $1/l$. This normalizes the largest absolute eigenvalues of all tensors to 1. We allow the user to rescale the tensor according to a user specified contrast parameter n as follows:

$$T_r = T^n \tag{8}$$

where T and T_r are the original and rescaled tensors respectively. The exponentiation operation is that defined on matrices. For a general tensor T with eigenvector matrix E and diagonal eigenvalue matrix $diag(\lambda_1, \lambda_2, \lambda_3)$, the above equation can also be written as:

$$T_r = E \cdot diag(\lambda_1^n, \lambda_2^n, \lambda_3^n) \cdot E^{-1} \tag{9}$$

where the contrast parameter n can be any real number. The eigenvector matrix E is constructed by columns and all eigenvector are normalized to unity.

3.2 Basic Idea

Prior to visualizing the symmetric tensor fields, they are first decomposed into orthogonal eigenvector fields: $E = \{e_1, e_2, e_3\}$. The corresponding eigenvalues are: $\Lambda = \{\lambda_1, \lambda_2, \lambda_2\}$.

The basic idea of the 2D HyperLIC algorithm is as follows: given a 2D symmetric tensor field and an input noise field, a geometric primitive is placed over each location. This primitive is going to be deformed by the tensor field. Using the deformed primitive at each location, the noise texture values under each deformed primitive are averaged together to give the pixel value of the resulting image. This procedure is similar to the DDA algorithm described in [Cabral and Leedom 1993] where the noise texture under each vector line glyph is used to calculate pixel intensities. When this process is carried out over tensor fields (as opposed to vector fields), the anisotropic properties of the underlying tensor field are revealed.

We experimented with two types of geometric primitives. The first type is a circle. In general, tensors deform circles to ellipses. An advantage of this primitive is that we do not need to decompose the tensor into component eigenvectors. Circles work best in regions with isotropic tensors. In areas with linear tensors, the resulting textures have very sharp contrast which tend to obscure the orientation of the tensors in the local neighborhood.

The second type of primitive is a square, oriented over each tensor so that the sides are aligned with the orthogonal eigenvectors – hence the need to first find the eigenvectors. The tensor deformation of these eigenvector aligned squares results in rectangles as shown in Figure 1. The scaling factors are simply the eigenvalues. Since we have normalized and rescaled the tensor so that its largest absolute eigenvalue is always 1, the longest side of the sampling strip is always a constant. We choose the size of the squares as $1/25$ of the image dimension.

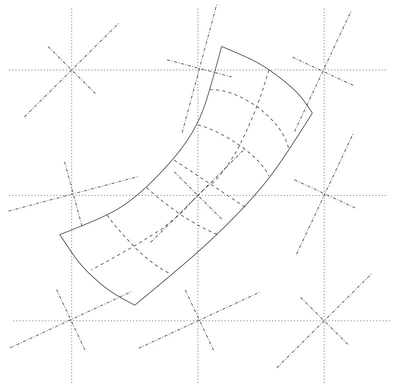


Figure 2: Sampling strip defined by a primary hyperstreamline and a set of orthogonal hyperstreamlines.

In regions with purely linear tensors, the square primitives are reduced to line segments and hence produce results similar to the DDA algorithm. In regions with isotropic tensors, HyperLIC is reduced to a smoothing algorithm, which averages the input texture within a local region. It produces a blurred image without sense of directions. In summary, HyperLIC maps the anisotropic properties of tensors to sharp and blurred regions in the resulting image.

3.3 Conceptual Algorithm

In LIC, the DDA is carried one step further, where the noise texture is integrated over the streamline rather than just a straight line segment [Cabral and Leedom 1993]. This allows one to capture more subtle features and improve contrast. This enhancement can be carried out with HyperLIC in the following fashion: One of the eigenvector fields is selected to generate a hyperstreamline using N integration steps. This hyperstreamline acts as a skeleton of the texture sampling region. Next, we draw hyperstreamlines using the other eigenvector at each of the N points along the first hyperstreamline. As shown in Figure 2, these two steps create a sampling strip along the first hyperstreamline.

Where the tensors are highly linear, the sampling strip is essentially the primary hyperstreamline and HyperLIC reduces to LIC. With isotropic tensors, this sampling strip forms a square which acts a smoothing filter on the input image. This improved algorithm works better than its basic version just as LIC works better than the DDA version.

3.4 Multi-pass Approach

While the conceptual algorithm captures subtle details of the underlying tensor fields, it is also much more expensive than LIC. This is because for each pixel, HyperLIC needs to create a 2D integration area as opposed to 1D in LIC. Here, we describe a multi-pass approach that accelerates the computation of HyperLIC.

Let P be a point in the tensor field, I_n be the input noise texture image, and I_o be the output HyperLIC image. Then, an output pixel is defined as:

$$I_o(P) = \frac{\sum_{i=-N}^{N}\sum_{j=-N}^{N} k(i,j) I_n(P_{i,j})}{\sum_{i=-N}^{N}\sum_{j=-N}^{N} k(i,j)} \quad (10)$$

$$P_{i,j} = P_{i-1,j} + \lambda_1(P_{i-1,j}) e_1(P_{i-1,j}) \Delta t \quad (11)$$

$$P_{0,j} = P_{0,j-1} + \lambda_2(P_{0,j-1}) e_2(P_{0,j-1}) \Delta t \quad (12)$$

$$P_{0,0} = P \quad (13)$$

where $\lambda_n(X),\ e_n(X),\ k(i,j),\ n = 1,2$ are the nth eigenvalues, eigenvectors and the weight function at point X. Δt is the integration step. In our experiment, the joint kernel function $k(i,j)$ is defined as the product of two kernel functions $k_1(i)$ and $k_2(j)$. k_1 and k_2 are constant for static images, and are defined in Equation 20 for animation sequences.

If we define $I_1(P)$ as:

$$I_1(P) = \frac{\sum_{j=-N}^{N} k_2(j) I_n(P_j)}{\sum_{j=-N}^{N} k_2(j)} \quad (14)$$

$$P_j = P_{j-1} + \lambda_2(P_{j-1}) e_2(P_{j-1}) \Delta t \quad (15)$$

$$P_0 = P \quad (16)$$

then, $I_o(P)$ can also be expressed as:

$$I_o(P) = \frac{\sum_{i=-N}^{N} k_1(i) I_1(P_i)}{\sum_{i=-N}^{N} k_1(i)} \quad (17)$$

$$P_i = P_{i-1} + \lambda_1(P_{i-1}) e_1(P_{i-1}) \Delta t \quad (18)$$

$$P_0 = P \quad (19)$$

Equations 14 and 17 are essentially the output images of the unnormalized LIC on $\lambda_2 e_2$ and $\lambda_1 e_1$ vector fields with input images I_n and I_1 respectively. Thus, we can reduce HyperLIC to a multi-pass unnormalized LIC. In the first pass, we apply the unnormalized LIC on $\lambda_2 e_2$ using the input image I_n and kernel k_2 to get an intermediate image I_1. In the second pass, we use the output image from the previous pass, I_1, as the input image and apply unnormalized LIC on $\lambda_1 e_1$ with kernel k_1 to generate the output image I_o.

Such a two-pass approach greatly reduces the amount of redundant computation in the conceptual algorithm. In each pass, only a 1D integration is needed. To further improve performance, FastLIC is implemented in each pass. The cost of HyperLIC on 2D tensors is only twice the cost of standard LIC. This allows one to interactively explore 2D tensor fields using HyperLIC.

In the vicinity of all tensors, the first pass filters the noise texture into a LIC-like image with dense and sharp lines, if we process the large eigenvalues first. In the second pass, the intermediate image remains unchanged if the orthogonal eigenvalues are small, implying a region of high anisotropy. If the eigenvalues are very high, which means low anisotropy, it blurs out the image across the orthogonal direction. It achieves the same goal as the conceptual algorithm in a different but much faster way.

We note that the conceptual algorithm and the multi-pass improvement both depend on which eigenvector is processed first. Theoretically, the order in which the eigenvector fields are processed will affect the final image. In practice, the differences are not noticeable (see Figure 4).

The differences are due to the spatial variation in the tensors. That is, following the major eigenvector for X steps from P and then switching to the minor eigenvector for Y steps may end up in a different location than following the minor for Y steps and then switching to the major for X steps. In symmetric tensors, the error is on the order of $O(L)$. In our experiments, L is set to $1/25$ of the size of images. Since L is relatively small for each point, the difference is negligible both theoretically and empirically. It means we can interpret the results from HyperLIC to be affected only by the underlying tensor field and not by the sequence of how the eigenvectors were processed.

Although the sequence of how the eigenvectors are processed is relatively unimportant, how the eigenvectors are classified dramatically affects the quality of the output images. That is to say, eigenvectors must be classified into major or minor fields. This algorithm produces nice results in most regions. However, because the major and minor eigenvectors may switch near critical points in the tensor field, integrating along an eigenvector field generates sudden transitions in an otherwise smooth tensor.

To avoid this problem, we label the eigenvector fields as follows: First, we find a point with high anisotropy and label its eigenvectors as first and second. Next, we iteratively label its neighbors consistently until all the points are labeled. An important point is that we set the eigenvectors as unchanged through the critical points. This labeling algorithm produces smoothly changing eigenvectors. For a smooth tensor field, there is no sudden change of major or minor eigenvectors. After we label the eigenvectors, the order of their processing is not important as we show in the results.

3.5 Sign Indeterminacy

HyperLIC images show the orientation of anisotropy but it doesn't show the direction of the major eigenvector. This is because the sign of the eigenvector is indeterminate. However, we can impose a consistent direction on the eigenvectors by using the signs of the eigenvalues. The idea is inspired by the behavior of charged molecules. A positively charged molecule M in an electric field dispels all other positively charged molecules and attracts all other negatively charged molecules. In other words, we can understand the sign of each molecule in this field by observing their motion relative to M.

To generalize this idea, we first synthesize a simple interrogational vector field. We then make the eigenvector directions follow this synthetic vector field. The synthetic vector field we chose in our experiment is $v(x,P) = x - P$, where P is a user-specified attracting point and x is a location in the vector field. The sign of an eigenvector e_i is chosen to make $\lambda_i e_i \cdot v(x,P) > 0$. With this direction-deciding algorithm, all eigenvectors with positive eigenvalues flow away from the attracting point while all eigenvectors with negative eigenvalues flow to the attracting point. We can easily identify the signs of the eigenvalues by observing whether the flow direction is toward or away from the attracting point.

In our experiments, a single attracting point is enough. However, for more complicated data, we may need multiple attracting points or a more complex interrogational vector field, or move the attracting point interactively.

3.6 Animation

After deciding the directions of eigenvectors, we propose an animation technique similar to that introduced in [Cabral and Leedom 1993] by varying the kernel functions according to time T.

$$k_n(w,T) = \frac{1+\cos(cw)}{2} \cdot \frac{1+\cos(dw+\beta T)}{2}, n = (1,2) \quad (20)$$

$$k(w_1,w_2,T) = k_1(w_1,T) \cdot k_2(w_2,T) \quad (21)$$

where c and d are two constants and β is the phase shift of the ripple function. $k_n(w), n = 1,2$ are kernel functions defined on each of the eigenvectors. The joint kernel function $k(w_1,w_2)$ is defined as the product of $k_1(w_1)$ and $k_2(w_2)$.

This time varying kernel function constantly shifts toward the eigenvector directions. For a linear eigenvector, only the kernel function for the major eigenvector has any effect, so HyperLIC produces a LIC-like animation flow. For an isotropic eigenvector, the kernel function is a multiplication of two simultaneously shifting kernel functions, so the flow pattern appears confused and ambiguous.

3.7 Extending HyperLIC to 3D

The HyperLIC algorithm described thus far works well in 2D. It shows the anisotropic properties of the tensors as varying intensities in the output image and maps tensor magnitude to color. The signs of eigenvalues are mapped to the directions of eigenvectors and visualized through animations. Although all the information of 2D symmetric tensors are visualized, the extra information in the third dimension are lost during tensor projection.

Fortunately, HyperLIC can be easily extended to 3D. The conceptual 3D HyperLIC works by averaging the input 3D noise texture using a 3D sampling volume defined by the three local eigenvectors. The two-pass 2D HyperLIC is extended to three passes in 3D. During each pass, one set of eigenvectors is used to apply unnormalized LIC on the input volume with the corresponding kernel functions. The output volume is then passed as the input volume in the next pass. The final 3D texture volume is then rendered using direct volume rendering.

We use a 3D diffusion tensor MRI brain data to demonstrate 3D HyperLIC. In medical data, fiber traces are important features. Fibers are represented by tensors with high anisotropy. On the other hand, tensors associated with white matter are relatively isotropic. HyperLIC visualizes the fibers as sharp lines and the rest of the brain as blurred regions. The global structure is insensitive to noise in the underlying data because HyperLIC smears out the local noise. To highlight the fibrous regions, we chose to map the local variance in the output volume to transparency prior to volume rendering using the following:

$$var(P) = \sqrt{\frac{\sum (I_o(P) - I_o(P_n))^2}{6}} \tag{22}$$

where P_n are neighboring points around P. This form of variance gives a measure of how similar or dissimilar neighboring points are from the value at P. This measure is low in isotropic regions and high in fibrous regions.

Using a transfer function that maps this variance to transparency, 3D HyperLIC can extract the fibrous regions of the brain data set. In this transfer function, the higher variances are mapped to lower transparency. In our experiments, we use a step function that maps all variances larger than 0.065 to the opacity of 0.1 and all others to 0. By choosing the right parameter, HyperLIC volumetric texture reveals the "edge" between the smooth and the sharp region. Tensor magnitudes are still mapped to hue as before while the HyperLIC volumetric texture is mapped to value in an HSV color model.

Accurate and smooth shading is crucial to visual perception. In traditional volume rendering algorithms, the normal of the surfaces are mapped to the gradient of data values. We find this to produce very noisy gradients. Instead, we apply a smoothing algorithm repeatedly to obtain a smoother normal. After each step, the cells are assigned the average of the old values in its neighboring cells. Note that we do not apply this smoothing algorithm on I_0 because it will smooth out the variance, which is critical to the extraction of the features. After smoothing out the variances during the gradient generation stage, we produce a more continuous shading on the same sharp features. The result is smoother shading on the fiber structures which greatly enhances the depth perception of the brain. An important point to note is that smoothed variances are only used to calculate the gradient for shading purposes. The transparency is still mapped to the original variances. Hence, what we are seeing accurately reflects what is in the data.

4 IMPLEMENTATION ISSUES

4.1 Inverse HyperLIC

The default HyperLIC algorithm shows strong directional information about the major eigenvectors, but it does not produce a strong visual effect about the other components. To compensate, we can switch the tensors in the preprocessing stage by swapping the tensors – so the minor eigenvectors become the major eigenvectors and vice versa. As a result, the images strongly resemble the minor hyperstreamlines (see Figure 3(c)).

4.2 Storage Requirement

3D HyperLIC is computationally expensive and requires a lot of memory. For a high resolution data set, 512^3 volume of floats in our experiment, the input and output volumes take up to 800 Mb of memory. To handle this huge memory demand, we solve the problem one layer at a time.

HyperLIC is a locally based algorithm. A voxel in the output volume is only affected by voxels within a certain distance of the corresponding input voxel. Because the eigenvalues are normalized, the maximum radius of the effective area is $2N + 1$ where N is the integration length. So, when we compute the output volume for layer Y, only layers from $Y - N$ to $Y + N$ in input volume are needed. We implemented this method with an output buffer of one layer, and an input buffer of up to $2N + 1$ layers. During each step, we update the output layer with the input layers. At the end of each step, we write the output layer into a temporary file and update the next layer of the input buffer, then go to the next step.

The results of the output volume after each pass in HyperLIC is stored in a temporary file. After each pass, we use the output file in the previous pass as input in the next pass. We implement this algorithm in parallel, by computing each output layer separately. Because the input textures are read-only, they are shared by all the rendering threads. When a rendering thread for an output layer is started, only the input layer not already in memory is loaded. This strategy reduces the memory requirements and makes the implementation very amenable to parallel computation on machines with multiple processors.

4.3 Rendering

3D HyperLIC produces a colored volume. In our experiment, this volume is 512^3. Interactive exploration of this large volume is a key component to understand the structure of the tensor data.

We employ a hardware-accelerated shell rendering algorithm to render this volume. Before the rendering, three sets of volume textures are generated, each of them is sliced along one of the three texture coordinates. When rendering from a given viewpoint, we pick the set of texture associated with the axis that best faces the camera. This is determined by computing the dot product of all three candidate axes and the camera and then use the one with maximum absolute value. If there is no clear winner, the output image, I_{out} is a weighted blend of three images. Let V_w be the camera orientation, $N_i,\ (i = 1, 2, 3)$ be the three axes, L_i be the length of side of a cell, I_i is the image rendered using texture slices along axis i. The output image is generated as follows:

$$H_i = \frac{\|V_w \cdot N_i\|}{L_i}, (i = 1, 2, 3) \tag{23}$$

$$M_i = \frac{H_i}{\sum_{i=1}^{3} H_i} \tag{24}$$

$$W_i = \begin{cases} M_i - \alpha, & M_i \geq \alpha \\ 0, & M_i < \alpha \end{cases} \tag{25}$$

$$I_{out} = \frac{\sum_{i=1}^{3} W_i \cdot I_i}{\sum_{i=1}^{3} W_i} \tag{26}$$

where α is a threshold parameter to choose between the smoothness and accuracy of the rendering. In our experiments, α is set as 0.3. The unoptimized implementation generates the 512^3 volume texture on the order of five minutes, and renders each image on the order of three seconds on a Dell Dimension 8100 with a single 1.5 GHz Pentium 4, 1 Gb of memory, and an nVidia GeForce2 Ultra.

5 RESULTS

In this section, we present results from HyperLIC on different data sets. The first data we experimented with is the single point load data. This stress tensor data is simple and thoroughly studied and therefore an excellent data for verifying the algorithm. The second data set is strain tensors derived from the flow past a cylinder with a hemispherical cap. This data is more complicated but it has also been used to demonstrate other published tensor visualization methods. The third data set is diffusion tensor MRI of the brain. Neural pathways are represented by tensors with high anisotropies. We present results both in 2D projection and 3D for this data.

Figure 3 includes 2D HyperLIC results on the single point load stress tensors from different viewpoints. Figure 3(a) is a slice from the middle of the volume and viewed from the point load direction. It is mostly composed of components from medium or minor eigenvectors. We see that the center of this slice is quite isotropic. Around the center is a ring formed by lines, which means tensors are highly anisotropic. It is the boundary where the minor eigenvalues are zero. From the animation available from the url provided in Section 1, we see flow within the ring going towards the center, while flow outside the ring going away from the center. This reveals that stress inside the ring is tensile, outside the ring is compressive, and the ring itself is free of stress away or toward the center. A little further away from the ring, we find another isotropic area. After the yellow area, the tensors are mostly represented by dense and sharp lines oriented radially that show high anisotropy in the regions.

Figures 3(b) and 3(c) are side views. Figure 3(b) contains mostly sharp lines, so eigenvalues are very large in these orientations. An interesting feature appears near the surface shown in more detail in Figure 3(d). There we see a change of pattern. The sharp lines change directions rapidly in a very narrow strip. Further observation in the animation reveals that flow near the surface is attracted to the point load, while flow further away is repelled. It clearly shows that the stress near the surface is tensile, while it is compressive in other areas. This can be confirmed in the stress tensor equation and is hardly shown by other visualization methods. Figure 3(c) is the same view as Figure 3(b), but shows the inverse HyperLIC which highlights the minor eigenvectors.

Images in Figure 5 are two results from strain tensors in the flow past a cylinder with a hemispherical cap. Figure 5(a) is from the inner layer (closer to the geometry) and Figure 5(b) is from the middle layer (farther away from the geometry). From these two images, we can clearly see the tensor orientations on the cap as well as locations of blurred isotropic tensors. We can also observe two degenerate wedge points. One is on the cylinder, while the other is on the cap.

Images from Figure 6 are from brain tensor data with different scaling parameters. Tensor normalization and rescaling is used in the preprocessing stage of HyperLIC to vary the degree and contrast in which anisotropy is depicted. The higher the scaling parameter, the more linear HyperLIC looks. In Figure 6(a), most regions except a few are blurred. In Figure 6(b), more fibers are visible. We can observe the main fiber structure in this picture. This demonstrates that tensor scaling allows the user to vary the amount of detail in HyperLIC images.

Figure 7 shows results from the brain tensor data using 3D HyperLIC. Images are from different viewpoints and rendered with shading. The light is always from the camera position. The main structures of the brain can be identified from these views. Figure 8 illustrates the effect of rendering without and with shading. Finally, in Figure 8(c), we use chromadepth mapping [Bailey and Clark 1998] to further enhance the depth perception with the aid of some inexpensive lens.

6 CONCLUSIONS

We presented a new technique for visualizing the anisotropy in symmetric 2D and 3D tensor fields. The technique provides a global, continuous representation of the field requiring minimal user input. Furthermore, the technique can produce animations that enhances perception and our understanding of the tensor field.

ACKNOWLEDGEMENTS

The brain dataset is courtesy of Gordon Kindlmann from the Scientific Computing and Imaging Institute, University of Utah, and Andrew Alexander from the W. M. Keck Laboratory for Functional Brain Imaging and Behavior, University of Wisconsin-Madison. The flow data is courtesy of NASA. This work is supported in part by NSF ACI-9908881 and NASA Cooperative Agreement NCC2-1260. Thanks go to Carol Mullane for help with proofreading.

References

BAILEY, M., AND CLARK, D. 1998. Using ChromaDepth to obtain inexpensive single-image stereovision for scientific visualization. *Journal of Graphics Tools 3*, 3, 1–9.

CABRAL, B., AND LEEDOM, L. 1993. Imaging vector fields using line integral convolution. *Computer Graphics Siggraph Proceedings*, 263–270.

DELMARCELLE, T., AND HESSELINK, L. 1993. Visualizing second-order tensor fields with hyperstreamlines. *IEEE Computer Graphics and Applications 13*, 4 (July), 25–33.

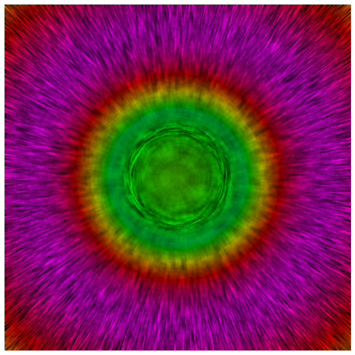
INTERRANTE, V., AND GROSCH, C. 1997. Statagies for effectively visualizing 3D flow with volume lic. In *Proceedings IEEE Visualization '97*, IEEE Computer Society Press, R. Yagel and H. Hagen, Eds., 421–424. Case Study - Flow Visualization.

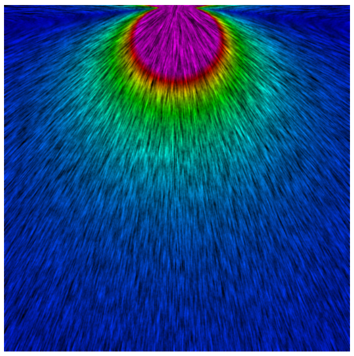
KINDLMANN, G. L., AND WEINSTEIN, D. M. 1999. Hue-balls and littensors for direct volume rendering of diffusion tensor fields. In *IEEE Visualization*, 183–189.

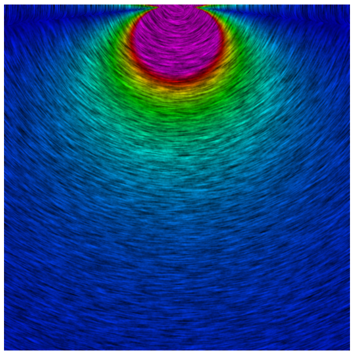
KINDLMANN, G., WEINSTEIN, D., AND HART, D. 2000. Strategies for direct volume rendering of diffusion tensor fields. *Visualization and Computer Graphics 6*, 2, 124–138.

SIGFRIDSSON, A., EBBERS, T., HEIBERG, E., AND WIGSTROM, L. 2002. Tensor field visualization using adaptive filtering of noise fields combined with glyph rendering. In *Proceedings of Visualization 02*, 371–378.

ZHUKOV, L., AND BARR, A. 2002. Oriented tensor reconstruction: Tracing neural pathways from diffusion tensor MRI. In *Proceedings of Visualization 02*, 387–394.


(a) Top view


(b) Side view


(c) Side view using Inverse HyperLIC

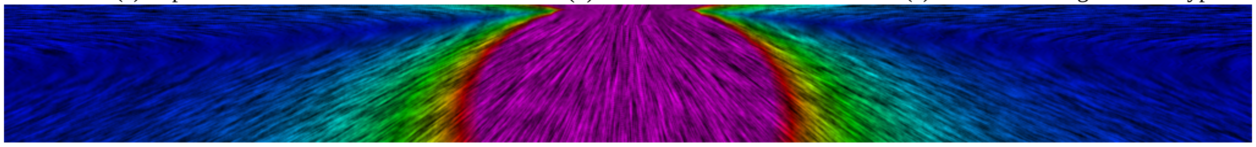
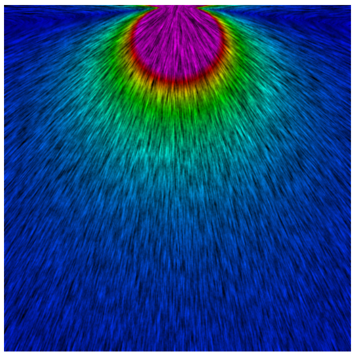

(d) Zoomed view of the top portion of (b)

Figure 3: Single point load data from different view points.


(a) Eigenvector order switched

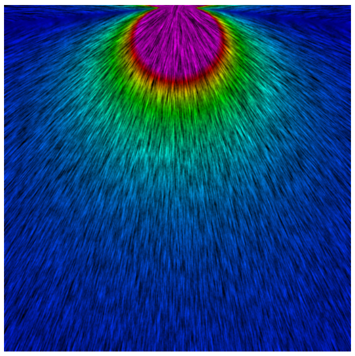
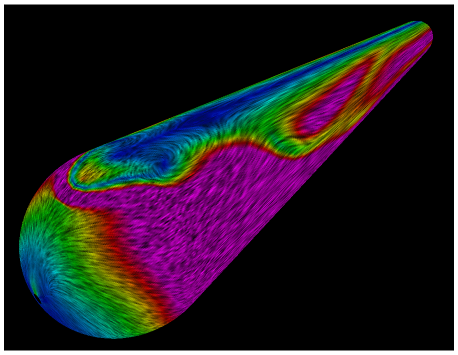

(b) Conceptual algorithm

Figure 4: There are no noticeable visual differences among the multi-pass algorithm in Figure 3(b) and (a) where the minor eigenvector is processed first, and (b) using the more expensive conceptual algorithm.


(a) Inner layer

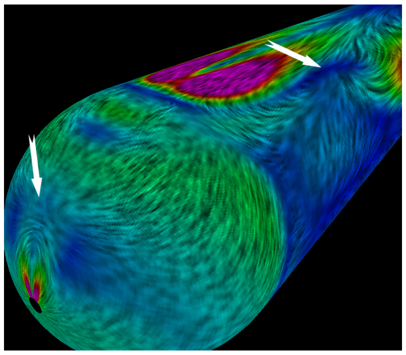
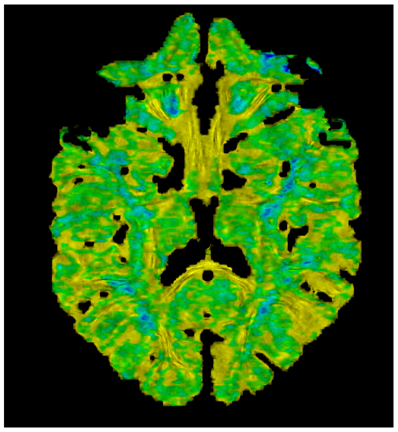

(b) Middle layer

Figure 5: Flow past a cylinder with hemispherical cap. HyperLIC of two different computational layers of the strain rate tensor. Arrows point to locations of degenerate wedge points.


(a) Scaling with $n = 2$

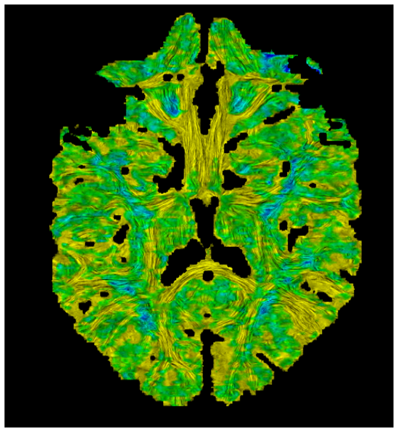
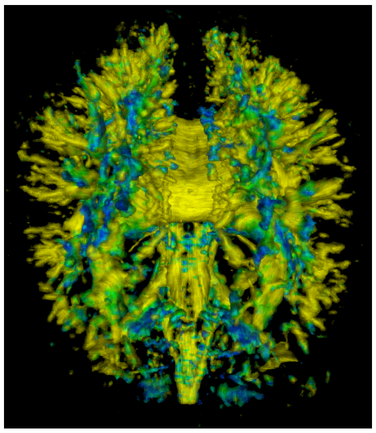
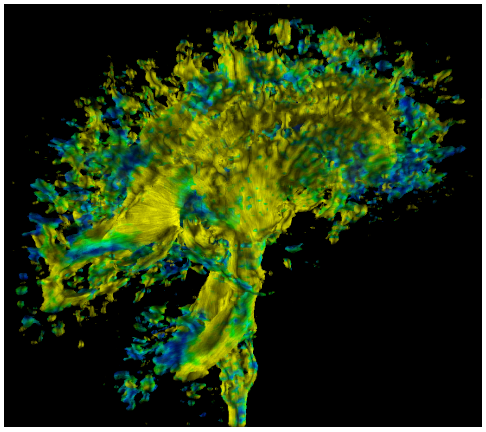

(b) Scaling with $n = 8$

Figure 6: A 2D slice of the brain data using different tensor scaling parameters.


(a) Back view


(b) Right view

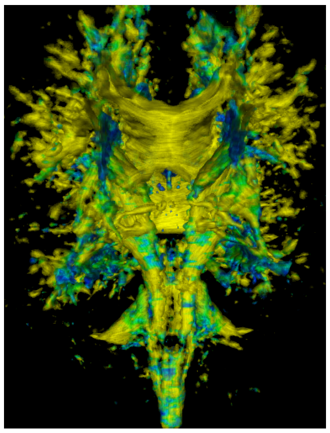
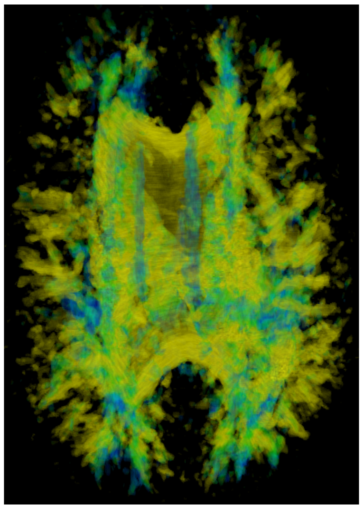
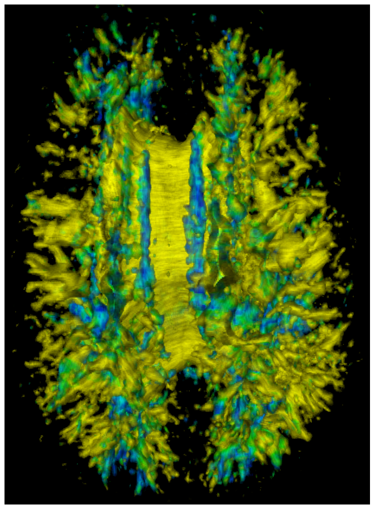

(c) Front view

Figure 7: 3D HyperLIC on diffusion tensor MRI brain data from different view points.


(a) Unshaded


(b) Shaded

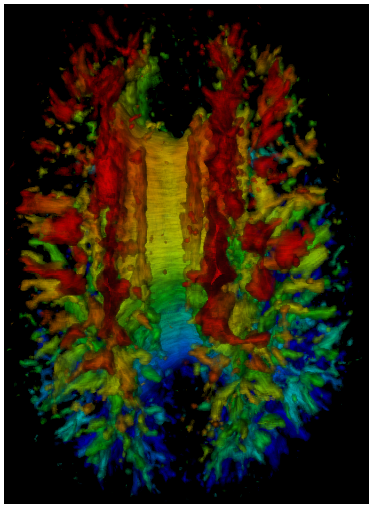

(c) Chromadepth

Figure 8: Comparison of different rendering techniques.