MESHING OF DIFFUSION SURFACES FOR POINT-BASED TENSOR FIELD VISUALIZATION

Ralf Sondershaus, Stefan Gumhold

WSI/GRIS, University of Tübingen, Germany, {sondershaus/gumhold}@gris.uni-tuebingen.de

ABSTRACT

The visualization of 3D vector and tensor fields in a 2D image is challenging because the large amount of information will either be mixed during projection to 2D or lead to severe occlusion problems.
In this work we segment from the symmetric 3D tensor field regions dominated by stream tubes and regions dominated by diffusion surfaces. The diffusion surfaces are integrated with a higher order Runge–Kutta scheme and approximated with a triangle mesh. Our main contribution is to steer the integration with a face-based coding scheme, that allows direct compression of the integrated diffusion surfaces and ensures that diffusion surfaces of any topology can be created.
Finally we sample the stream tubes and diffusion surfaces with points. The points from different entities are colored with different colors. We lit the points during rendering with a lighting model adapted to the tensor field. The resulting visualization of symmetric 3D tensor fields is sparse because of the sampling on points and allows for a deeper view inside the volumetric tensor field but also allows the simultaneous visualization of a dense set of tubes and surfaces.

Keywords: Tensor Field, Surface Integration, Surface Meshing, Visualization, Point Rendering, Diffusion Surfaces

1. INTRODUCTION

The visualization of 3D vector and tensor fields in a 2D image is challenging because the large amount of information will either be mixed during projection to 2D or lead to severe occlusion problems.

A lot of work has been done to visualize vector fields. Stream lines and stream surfaces are popular visualization techniques. A stream line is a curve where for every point on the curve the associated vector is tangent to the curve. One can imagine a stream line as the path that a particle takes through the vector field. Stream lines do not intersect each other except for points where the vector field vanishes or is undefined.

A stream surface is the path that a curve takes through the vector field and can be thought of as the dense collection of stream lines, all starting at a given curve.

The situation changes slightly if we look at symmetric 3D tensor fields. Throughout this paper we use the term tensor for symmetric 3D tensors. Symmetric 3D tensors play a great role in physics or medicine as for example diffusion tensors are symmetric 3D tensors. At every point a tensor field contains a (symmetric) tensor, i.e. a symmtric 3×3-matrix, instead of a single vector.

Eigenvector decomposition of the tensor is a popular approach to analyze a tensor. A symmetric tensor can always be decomposed into a diagonal matrix Λ with the three eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ on the diagonal and an orthonormal rotation matrix V with $VV^t = \mathbf{1}$, with the unit matrix $\mathbf{1}$:

$$\begin{array}{rcl} \forall T \in \mathbf{R}^{3\times 3} & \text{with} & T = T^t : \\ \exists V, \Lambda \in \mathbf{R}^{3\times 3} & \text{with} & VV^t = \mathbf{1}, \Lambda_{ij} = 0 \Leftarrow i \neq j : \\ & & T = V \Lambda V^t. \end{array}$$

The columns $v_i \overset{def}{=} V_{.i}$ of V form an orthonormal basis of $\mathbf{R}^3$ and are called the eigenvectors. The combination of eigenvectors and eigenvalues (V, λ) is called the eigensystem of the tensor. If a unit sphere is scaled in the direction of the eigenvectors with the eigenvalues we obtain an ellipsoid that can be used to visualize the tensor. In the case of diffusion tensors the ellipsoids describe for any possible direction the rate of diffusion. Particles would have to be traced in all possible directions with speeds given by the ellipsoid. A diffusion tensor can be imagined as a description of how a spherical water drop is diffused into an ellipsoid. The terms stream line and stream surface can therefore not so easily applied to tensor fields.

Diffusion tensors often degenerate over large regions, such that their ellipsoids look like cigars or like pancakes. In the case of a cigar we speak of linear anisotropy, when one eigenvalue dominates the other two. In the other case we speak of planar anisotropy and two eigenvalues are much larger than the last one. If all eigenvalues are of similar size the corresponding region of the tensor field is called isotropic.

The eigenvector with the largest eigenvalue is called the major eigenvector, the eigenvector with the smallest eigenvalue is called the minor eigenvector, and the eigenvector with the medium eigenvalue is the medium eigenvector.

Given the anisotropy of tensors, the domain of a tensor field can be partitioned into linear, planar, and isotropic regions. Every region only contains tensors of the one specific type of anisotropy.

Within linear regions, a tensor field can be interpreted as a vector field formed by the major eigenvectors. Thus, we can define stream tubes as tubes whose middle axis is a stream line. Every point on the stream line is tangential to the major eigenvector of the tensor at this point. The cross section of the tube is defined by the medium and minor eigenvectors which are perpendicular to the major eigenvector.

Within planar regions, a diffusion surface can be defined as surface whose tangential plane for every point is the plane defined by the major and medium eigenvectors, i.e. it is normal to the vector field of the minor eigenvectors. We use the term diffusion surface here because the term stream surface may be misleading to what a stream surface is for a vector field.

We segment from the symmetric 3D tensor field regions dominated by stream tubes and regions dominated by diffusion surfaces. We reconstruct a dense set of stream tubes and diffusion surfaces and point sample them in way that the distance between two points is inversely proportional to the diffusion rate. Thus the points are closely spaced along a stream tube and sparsely orthogonal to it. On a diffusion surface the points are closely spaced over the surface but the surfaces are further apart from each other. The human eye of the observer will automatically merge close points to tubes and surfaces. The points from different entities are distinguished by their color. We lit the points during rendering with a lighting model adapted to the tensor field. The resulting visualization of symmetric 3D tensor fields is sparse because of the sampling on points and allows for a deeper view inside the volumetric tensor field but allows on the other hand the simultaneous visualization of a dense set of entities.

2. RELATED WORK

Isosurfaces The marching cubes algorithm [1] can extract an isosurface from a scalar field. An isosurface is thereby defined as the collection of points with equal scalar values. The marching cubes algorithm creates a triangular mesh that approximates the isosurface. For every vertex of the triangular mesh, a normal is calculated to perform lighting during rendering.

The basic algorithm loops over all (small) cubes whose corner points are in the center of eight voxels of the discrete scalar field. For every such cube the isosurface is located within the cube. There are $2^8 = 256$ ways how the surface may intersect with the cube. Every case is triangulated. A surface intersects an edge of the cube when one vertex is outside and the other vertex is inside the surface.

Stream Surfaces Stream surfaces from vector fields can be represented as parametric surfaces. A simple approach to construct such parametric stream surfaces is to place a number of seed points onto the original curve, to trace these points along their stream lines, and to connect the resulting stream lines. Remember that stream lines of vector fields can never cross each other.

This approach has many drawbacks. For example, consider converging or diverging stream lines. Connecting diverging stream lines may result in slivers or even don't approximate the surface correctly because of the large area that is approximated linearly by the triangle.

Hultquist [2] improves this parametric stream surface approach. An initial set of stream lines is traced and as the tracing proceeds, additional stream lines are introduced in the case of diverging stream lines, or stream lines are removed on cases of converging stream lines.

Scheuermann et al. [3] presented an algorithm that is related to Hultquist. The stream surfaces are calculated for tetrahedral grids. A surface is propagated through a tetrahedra, calculating the intersections between the surface and the tetrahedra. The surface within a tetrahedra is a ruled surface, which means that the surface is generated by two stream lines that are blended by line segments. All calculations are done in barycentric coordinates of the tetrahedra which simplifies the formulas.

Van Wijk [4] models stream surfaces as implicit functions instead of parametric surfaces. A stream surface is given by the implicit function $f(x) = C$ where C is a (scalar) constant. The difficulty is to find the function f. Once f is found, a family of stream surfaces can be generated efficiently by varying C.

Van Wijk calculates f from the convection equation for incompressible flow with f as the transported quantity and $\vec{v}$ as the velocity.

$$\frac{\partial f}{\partial t} = -\nabla f \cdot \vec{v}$$

A range of values is placed along the boundary as initial (boundary) values. The convection equation is then solved over time using some numerical integrator until a steady state is reached. One can imagine this process as inserting varying concentrations of ink along the boundary of water. After some time, this ink is distributed (not necessary evenly!) within the water and the stream surfaces are the areas of equal concentration. Once the implicit function f is calculated, the isosurface for a particular value C is extracted and rendered with the marching–cubes algorithm.

In the case of diffusion surfaces the implicit approach is not possible. In the implicit approach the surface normal corresponds to the gradient ∇f of the implicit function. The diffusion surfaces are defined to be orthogonal to the vector field of the minor eigenvectors of a tensor field. Therefore, we are given ∇f and need to find f. From the Helmholtz–Hodge decomposition follows that this is possible only if the rotation of the vector field of the minor eigenvectors vanishes. As the rotation does not vanish for arbitrary symmetric tensor fields, the implicit approach is not possible. Figure 7 shows the diffusion surface of a tensor field with a planar region where the minor eigenvector field has quite a lot rotation.

Diffusion Surfaces There hasn't been much work done yet to extract diffusion surfaces from tensor fields. Zhang et al. [5] presented a technique to extract stream tubes and diffusion surfaces from volumetric diffusion tensor MR images. Stream tubes are extracted in linear regions and diffusion surfaces in planar regions. So stream tubes represent structures with primarily linear diffusion while diffusion surfaces represent structures with primarily planar diffusion. Additional information is encoded in the color and cross section of the stream tubes.

Our approach of extracting diffusion surfaces is similar to [5]. That's why this approach is discussed in more detail. Zhang et al. generate a dense set of stream tubes and diffusion surfaces and cull them later using a set of metrics.

Linear regions are interpreted as vector fields formed by the major eigenvectors of the tensors. Thus the trajectory of a stream tube is a stream line through this vector field. The MR images are interpolated using tricubic B-Splines to get tensors not only at the sample points of the MR images. Zhang et al. generate seed points for every sample point within a linear region and jitter them within the voxel. The stream tube starts at a seed point and follows the major eigenvector field both forward and backward. An second-order Runge-Kutta integrator is used to track the stream line.

Diffusion surfaces[1] are extracted from planar regions. The major and medium eigenvectors of a tensor at a point in space define the tangential plane of the surface at this point. Again, the seed points are placed into the voxels by jittering the sample points.

[1]Zhang et al. call the diffusion surfaces stream surfaces. We prefer the term diffusion surface to avoid confusion with the term stream surface from vector fields.

Starting from a seed point v, six initial search directions are distributed evenly around v. Every search direction is tracked and thus follows the shape of the surface. A triangle is created for every pair of neighboring edges. This first step creates a triangle fan consisting of six triangles.

From every vertex u new search directions are created by projecting the triangles that are adjacent to u onto the tangential plane $P(u)$ and the initial directions in $P(u)$ that are not covered by triangles. This is repeated for every newly generated vertex.

The new search directions of a vertex u are traced through a vector field which is defined as the linear combination of the normalized major and medium eigenvectors which lies on a Plane P_1 that is both perpendicular to the tangential plane $P(u)$ at u and contains the search direction.

The extension of the diffusion surface stops if it gets out of the data boundary, hits a low planar region, enters a region of low signal–to–noise ratio, or incurs a high curvature term.

While rendering, color is mapped onto the surface to represent the planar anisotropy.

3. POINT-BASED TENSOR FIELD VISUALIZATION

3.1 Volume Segmentation

As already mentioned in the introduction, the tensor field domain is partitioned into linear, planar, and isotropic regions. We use three quantities of a (diffusion) tensor to define this partition as suggested by [6]:

$$\begin{aligned} c_l &= \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3} \\ c_p &= \frac{2(\lambda_2 - \lambda_3)}{\lambda_1 + \lambda_2 + \lambda_3} \\ c_s &= \frac{3\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \end{aligned}$$

where c_l measures linear anisotropy, c_p planar anisotropy, and c_s isotropy. Note that $c_l + c_p + c_s = 1$. λ_i are the eigenvalues of the tensor with $\lambda_1 \geq \lambda_2 \geq \lambda_3$. The greater c_l is the more the ellipsoid looks like a cigar (and the smaller c_p and c_s are). Similarly, the greater c_p is the more the ellipsoid looks like a pancake. Finally, if c_s equals 1, the ellipsoid becomes a sphere (all eigenvalues are 1 and c_p and c_l are 0), see figure 1.

After this segmentation of the volume, we can trace stream lines in linear regions and diffusion surfaces in planar regions. We use thresholds on c_l and c_p to classify regions.

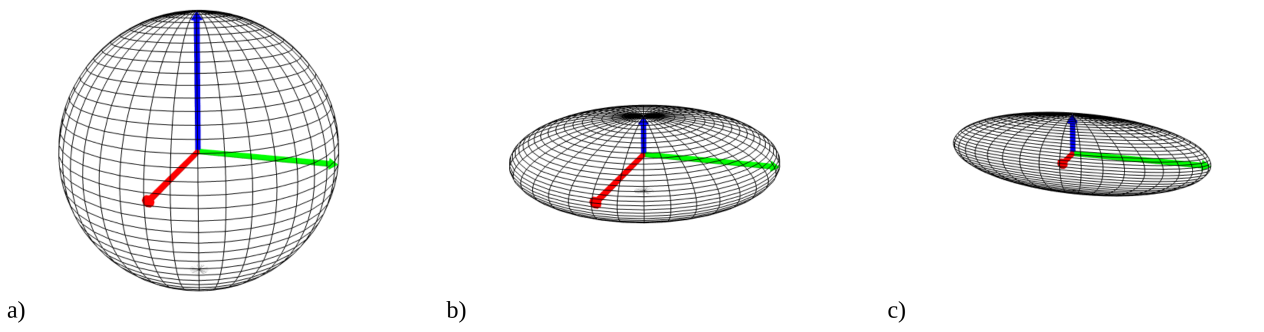

Figure 1: A tensor can be classified as being isotropic ($c_l = 0$, $c_p = 0$, $c_s = 1$), planar ($c_l = 0$, $c_p = 0.61$, $c_s = 0.39$), or linear ($c_l = 0.57$, $c_p = 0$, $c_s = 0.43$).

3.2 Distributing Points

We render stream tubes and diffusion surfaces as collections of points. Different colors are used to distinguish points from different entities. The tubes and surfaces shall be point sampled with the density described by the inverse of the diffusion rate given by the symmetric tensors.

Depending on the entity, that points were sampled from, the points are lit differently. Points from stream tubes are lit with the lighting model for lines as proposed by Zöckler et al. [7], whereas points on the diffusion surfaces are lit with the standard Blinn-Phong lighting model provided by OpenGL.

3.3 Stream tubes

Similar to [5] we look at the tensor field in linear regions as a vector field consisting of the major eigenvectors of the tensors. This vector field is traced.

We subdivide the volume into a regular grid which may be given automatically by the resolution of the image data. Otherwise, the resolution of the grid is specified explicitly.

To create stream tubes, we first create a stream line for every stream tube. This stream line is the trajectory of the latter stream tube.

We place a seed point into every grid cell. The tensor of this seed point needs to have linear anisotropy. We integrate a stream line starting at the seed point into both forward and backward direction using a second-order Runge-Kutta integrator [8]. A stream line consists of a list of ordered points p_i and is linearly approximated as a line segment between two successive points p_{i-1} and p_i.

We want to place the points along a stream line such that the density of the points on a stream line represents the linear anisotropy of the tensors involved as described above.

To realize this behavior, we control both the arc length of the integration process and the step width of the second-order Runge-Kutta integrator.

The integration of the stream line stops if any of these cases happens:

- Outside of data volume.
- Left region of linear anisotropy $c_l > C_l$ where C_l is the threshold for linear anisotropy.
- Extended point is ”too” close to a previously calculated point.

The third point is motivated by artificial datasets where a stream line may be a (closed) circle. In order to stop the integrating process, the integrator needs to check if it reaches a part of the stream line that was previously integrated. For a fast local access to the points and line segments of the stream line, we sort the extended points into an octree. The integrator just needs to look up the octree to find proximate points.

A stream tube follows the trajectory defined by the stream line. Our approach is to render the (extended) points of the stream line only instead of rendering the whole tube around the stream line.

3.4 Diffusion surfaces

A crucial point for our visualization technique is to distribute the points across diffusion surfaces. We want the points to be distributed according to the diffusion rate over the diffusion surface. The higher the diffusion rate, the closer have to be the points. Remember that we want to render points instead of shaded surfaces. The human eye connects points that lie closer together and therefore follows automatically the more likely diffusion direction.

Although we are only interested in rendering points, for the integration of the diffusion surface it is advantageous to also build a triangle mesh to ensure a proper diffusion surface. Furthermore the connectivity information allows for smoothing and successive remeshing steps.

Although the diffusion surfaces do not have to be isosurfaces of an implicit function, they have to be manifold for differentiable tensor fields. This can be easily shown from the definition. The diffusion surface is restricted to the subdomain of the tensor field, which is classified planar. Therefore, the minor eigenvector is defined everywhere and varies continuously over the planar subdomain as we supposed the tensor field to be differentiable. As the minor eigenvector uniquely defines the normal direction of the diffusion surface, the diffusion surface has to be manifold with border loops at the border of the planar domain.

4. DIFFUSION SURFACE INTEGRATION

As the diffusion surface has to be manifold with border, the integration process that approximates the diffusion surface via a triangle mesh is very similar to the encoding or rather decoding process of a faced-based compression scheme. We used exactly the same building scheme to build the triangular diffusion surface. There are three advantages with this approach. Firstly, we can re-use the minimum set of building operations that allow to create manifold meshes of arbitrary genus. Secondly, we can re-use the data structures used for face-based coding such that the implementation of the diffusion surface integrator becomes very simple. And the third advantage is that we can directly encode the triangular diffusion surface into a space efficient representation, such that we can easily build in-core a large number of diffusion surfaces of high resolution.

We used a face-based compression scheme similar to the cut-border machine [9] and the edge-breaker [10] (see also [11] for an introduction to face-based coding) for coding and to steer the integration. A short review of the method and the basic building operations are given in the next subsection.

4.1 Face-Based Coding of Triangle Meshes

Face-based coding techniques compress triangle meshes which consist of a list of vertices and a list of triangles, each triangle containing three vertex indices and the indices of the edge-adjacent triangles.

The schemes are based on a region growing traversal of the triangle mesh. The traversal begins for example with an arbitrary seed triangle. The border of the growing region is called the *cut-border*. It divides the mesh into the *inner* and the *outer part*, which contain the already processed and the untouched triangles respectively. Triangles are added to the inner part at a distinguished current cut-border edge which is called the *gate*. After each addition of a triangle the gate moves on to another cut-border edge, until all triangles of an edge-connected component of the triangle mesh have been compressed. This is done for each edge-connected component. The choice of the next gate location defines the *traversal order* and steers how the cut-border grows over the mesh.

The face-based coding scheme encodes a bit-code each time a new operation is added. The decoding performs the same traversal and builds the face according to the encoded oper-

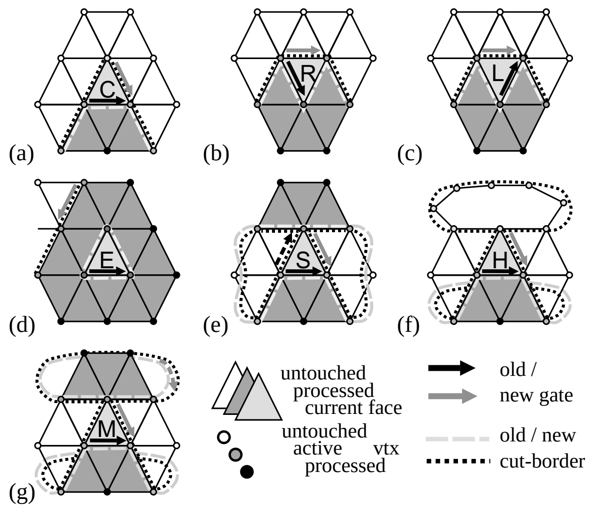

Figure 2: The different cut-border operations in which the processed triangle can be incident to the cut-border.

ation. The different possible operations by which the next triangle is incorporated into the inner part at the gate are illustrated in figure 2. The *center* operation C in (a) adds a new triangle to the growing region that is incident only to the old gate and to a new vertex. The gate is moved to the right newly added cut-border edge such that it cycles around its target vertex. The current face in the *right/left* operation R/L in (b/c) is incident to the gate *and* the next/previous edge on the cut-border. The neighborhood of the pivot vertex is closed and a new pivot vertex is chosen with the new gate location. In the *end* operation E in (d) all edges of the current face are incident to the cut-border and the cut-border closes. The other growing operations describe cases when the third vertex of the current face is on the cut-border. (e) shows the *split* operation S, where the cut-border grows into itself and is split into two loops with two gate locations. One cut-border loop is pushed onto a stack and processed after the other one is eliminated by an *end* operation. In order that the decoder can replay the split operation the position of the third vertex relative to the gate is encoded. The *hole* operation H (f) merges the current cut-border with a border loop. We will handle border loops in a different way as done by the cut-border machine [9]. We encode a border operation B, whenever the gate is a border edge of the mesh. As triangle meshes describe two dimensional surfaces in three dimensional space, two cut-border loops can grow together again, actually once for each handle of the triangle mesh. The operation which unifies two loops is called *merge* operation M (g). It merges the current cut-border loop with another cut-border loop and takes two indices, the index of the other cut-border loop and the location inside that other loop.

The cut-border data structure consists of a stack of doubly linked lists of cut-border edges. Each cut-border edge stores

```
Input: seed location $x$
integrate edge from $x$ to $y$
init cut-border to $(x, y)$
while cut-border not empty
    choose gate
    decide operation
    apply operation to cut-border
```

Figure 3: Structure of the integration algorithm.

the indices of its start vertex and of its adjacent triangle in the inner part. The initialization creates a cut-border with three edges. Each C,S or M operation inserts a cut-border edge after the gate. The R and L remove the next or previous cut-border edge and the E operation closes the loop on top of the stack.

4.2 Cut-Border Based Surface Construction

As the face-based coding scheme can encode any manifold mesh, one can also build any triangulated diffusion surface during the integration.

The input to our diffusion surface integrator is a seed location in a planar region of the volumetric domain of the tensor field. We integrate the diffusion surfaces with the face-based building operations C,L,R,E,S,M,B. Opposed to the face-based scheme we start the building process with a single edge and initialize the cut-border to a loop of two cut-border edges around the edge like the face-fixer proposed by Isenburg [12]. This first edge is created by integrating from the seed location in the direction of the major eigenvector of the tensor field. The integration of a new edge is described in the next subsection in detail.

The diffusion surface is built from the initial cut-border by extending it at one of the cut-border edges, which is called the gate. The choice of the gate steers how the cut-border grows the diffusion surface and is described in subsection 4.4 in detail.

We summarized the integration algorithm in figure 3. After the cut-border has been initialized from the seed location we successively select a gate edge, decide which of the operations C,L,R,E,S,M,B to perform and apply it to the cut-border until no cut-border edges are remaining. This can either happen, when the diffusion surfaces closes up or when all cut-border edges were transformed into border edges of the mesh by a B operation at the boundary of the planar domain of the seed location. Figure 4 illustrates the growing process for a spherical diffusion surface.

In this subsection it remains to explain how we decide for a given gate location, which operation has to be performed. We first propose one of the operations E,L,R,B or C and then check for E,L,R or C if we should not have performed an S or M instead.

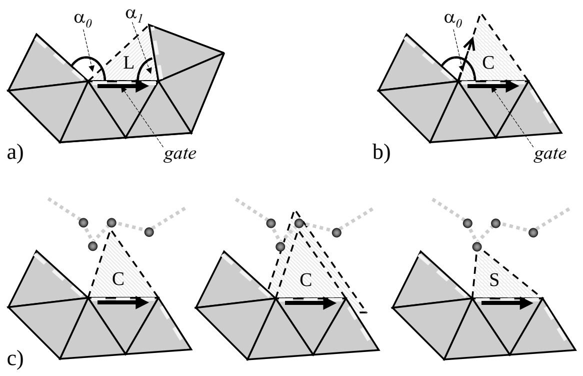

Figure 5: a) angles measured to decide for L, R or E operations b) integration direction c) check for split or union

First we decide if we should propose an L,R or E operation based on the exterior angles between the gate edge and the previous or next edge on the cut-border as depicted in Figure 5 a). If one of the angles α_0 or α_1 is smaller than a threshold angle, that we set to seventy degrees, we decide for an L or R operation. When the length of the current cut-border loop is only three, we decide for an E operation instead of L or R.

If both angles are larger than the threshold, we try to propose a B operation by trying to integrate the left edge of the new triangle as shown in Figure 5 b). The direction for integration should be in the planar case always sixty degrees. For curved surfaces this is not possible anymore. Here we chose to subdivide the angle α_0 into k equal parts such that the resulting angle is closest to sixty degrees. In the example of Figure 5 b) k is two such that we chose the direction $\alpha_0/2$ away from the gate.

The integration process is in the next subsection. It returns a target location or reports failure if the integration left the planar domain. In case of failure we decide for a border operation B, that will always be performed. Otherwise we can construct with the target point a new triangle labeled with C in the Figure 5 b).

Now we either perform a border operation or propose a new triangle added by E,L,R or C. This triangle can intersect a distant cut-border part as illustrated in Figure 5 c). Here we should rather perform a S or M operation with the vertex closest to the gate. Whether S or M is performed can be decided by the cut-border data structure from the vertex to which the gate will be connected. To find out if an S or M has to be performed, we entered all cut-border edges in an octree and checked for the proposed E,L,R and C triangles if they cover other cut-border edges. In order to also connect to close outside cut-border vertices we enlarge the proposed triangles by 30% before we checked in the octree. From the

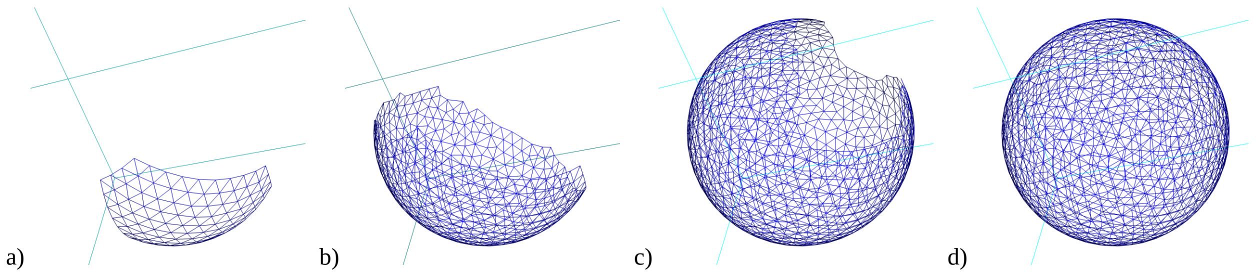

Figure 4: Illustration of how a diffusion surface is grown.

covered cut-border edges and vertices we selected the cut-border vertex closest to the center of the gate and proposed an S or M operation. See Figure 5 c) for an illustration of this process.

4.3 Integration of New Edges

For every search direction, we define the search plane as the plane that is orthogonal to the tangential plane at the start point, contains the search direction vector and the start point itself. This approach is similar to [5].

Every search plane defines a vector field along which we integrate from the start point. A vector of this vector field is, simply spoken, defined by the cut of the search plane and the plane which is spanned by the major and medium eigenvector of a tensor. The length of this vector is just the distance between the location of the tensor and the cut point of the search plane with the ellipse defined by the major and medium eigenvector, see figure 6.

To perform this operation quickly, we use the following simple mathematics. Let n be the normal of the current search plane, and T be the tensor matrix at the current point. The vector field is then defined by the following operations:

$$\begin{aligned} \tilde{n} &= T^t n \\ n_\perp &= \begin{pmatrix} -\tilde{n}_1 \\ \tilde{n}_0 \\ 0 \end{pmatrix} \\ v_t &= T n_\perp \end{aligned}$$

We transform the normal vector of the search plane into the coordinate system defined by the eigenvectors. That is just a matrix–vector multiplication. Because we want the vector to lie in the plane of the two largest eigenvectors, we project the vector into this plane by setting the last coordinate to 0. Then, an orthogonal vector is set up by multiplying the vector to a 90 rotational matrix. This can be done explicitly by just changing the coordinates. This orthogonal vector is then transformed back into the world coordinate system to be the vector of the vector field within the search plane.

The step width for the integration process of one search direction is adapted to the eigenvalues in order to distribute the

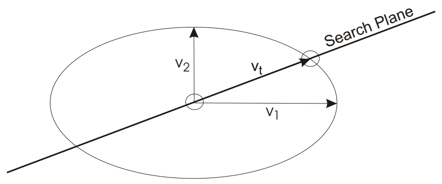

Figure 6: A vector of the vector field defined by the cut of the search plane and the ellipsoid within the plane of the two largest eigenvectors.

points according to the anisotropy of the tensor field

$$h = C\sqrt{\sum_i w_i^2 n_{\perp i}^2}$$

where $n_\perp$ is the vector orthogonal to the normal vector of the search plane as defined above, h is the step width, and C is a user-controlled constant which can further influence the density of the points. The weights w_i are chosen to represent the proportional behavior to the anisotropy as

$$w_i = \frac{1}{\lambda_i}$$

4.4 Traversal Order

In order to avoid a large number of S or M operations we used a similar strategy as proposed by Alliez and Desbrun [13]. The method is based on the observation that S operations arise very seldom at cut-border edges that are in a convex region. Therefore we measured for each cut-border edge the two exterior angles α_0 and α_1. The smaller these angles are, the more convex is the region around this cut-border edge. As it is already fine if one of the angles is small, we sorted the cut-border edges according to the minimum if the exterior edges into a priority queue. Each time a new gate has to be chosen, we extracted the most convex from the priority queue. After each basic operation we updated also the cut-border edges adjacent to newly added or removed ones and re-positioned them in the queue. Figure 4 illustrates that the cut-border stays nicely shaped during the integration process.

5. RESULTS & ANALYSIS

We tested our algorithm with tensor fields that include singularities which need to be bypassed by the integration and meshing process. We created artificial tensor fields to simulate different behavior.

The Spiral example has singularities on the z axis. The tensors have planar anisotropy around the z axis, see figure 7. The diffusion surfaces are rendered as triangular meshes to show how the mesh generation follows exactly the diffusion surface and thereby bypasses the singularities.

The Sphere dataset, figure 4 d, presents the ability to handle self–intersections. If the mesh grows into a previously integrated region, the algorithm detects this intersection and connects the boundaries to build a triangular mesh.

A dataset similar to sphere, see figure 11, demonstrates the usefulness and power of the point–rendering approach. The interior stream lines are completely enclosed by a mesh but they remain visible.

The Sinosoidal example shows both stream tubes and stream surfaces within a dataset that varies anisotropy very frequently. Figure 9 shows a simple preview of the dataset displaying the major, medium, and minor axes of the tensors within a 10x10x10 grid. The color encode the different anisotropies. Green is planar anisotropy, red linear, blue isotropy, and gray is undefined. Figure 9 shows the point–rendered stream tubes and diffusion surfaces.

The crucial part of the performance of our algorithm is the integration process itself. Thus the step width of the integration step is the limiting factor and needs to be adapted very carefully to the anisotropy of the tensor field.

Note that the distribution of the points over the diffusion surface may not be optimal. We did not implement an optimization algorithm for placing the points. We can ensure that the whole diffusion surface is extracted because our approach starts with an initial triangle and grows it's border until it reaches a non–linear region, and that the distance between points along edges that were integrated is optimal in terms of that this distance accords to the diffusion rate along this edge. But there may be edges that were not integrated but artificially inserted by the cut–border operations. The optimal distribution of points over a diffusion surface is topic of further research.

6. CONCLUSION

We presented an algorithm that tracks stream tubes and diffusion surfaces of tensor fields according to the anisotropy of the tensor field. The resulting points are rendered as point clouds where different entities are distinguished by color. Because the points of an entity are much closer than the distance between the single entities, the human eye is able to differentiate between these entities, additionally to the color coding. Furthermore, the point rendering enables a deeper insight into the volumetric information of the tensor field.

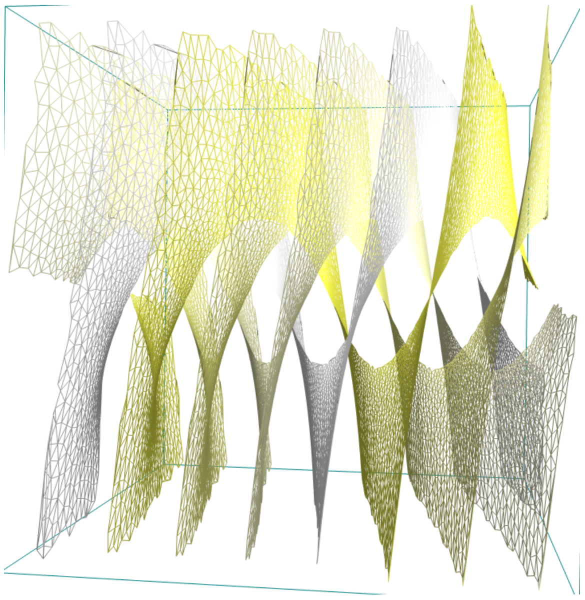

Figure 7: The diffusion surface is a spiral around the singularities at the z axis.

The generation of the diffusion surfaces is very similar to the decoding process of a faced-based compression scheme. Exactly the same building scheme can be used. This allows for a simple implementation and a compact representation of the generated mesh. The diffusion surface can thereby be of arbitrary genus.

References

[1] Lorenson W.E., Cline H.E. "Marching Cubes; A High Resolution 3D Surface Reconstruction Algorithm." *Computer Graphics (Proceedings of SIGGRAPH'87)*, vol. 21, pp. 163–169. Jul 1987

[2] Hultquist J.P. "Constructing Stream Surfaces in Steady 3D Vector Fields." RNR Technical Report RNR-92-025, NASA Advanced Supercomputing Division (NAS), Aug 1992. URL `http://www.nas.nasa.gov/Research/Reports/Techreports/1992/rnr-92-025-abstract.html`

[3] Scheuermann G., Bobach T., Hagen H., Mahrous K., Hamann B., Joy K.I., Kollmann W. "A tetrahedra-based stream surface algorithm." *IEEE Visualization 2001 Proceedings*, pp. 151–158. 2001

[4] van Wijk J.J. "Implicit stream surfaces." *Proceedings of the Visualization '93 Conference*, pp. 245–252. Oct 1993

[5] Zhang S., Demiralp C., Laidlaw D.H. "Visualizing Diffusion Tensor MR Images Using Streamtubes and Streamsurfaces." *IEEE Trans. Visualization Computer Graphics*, p. (in press), 2002

[6] Westin C.F., Peled S., Gubjartsson H., Kikinis R., Jolesz F. "Geometrical diffusion measures for MRI from tensor basis analysis." *Proceedings of ISMRM*. 1997

[7] Zöckler M., Stalling D., Hege H. "Interactive Visualization Of 3D-Vector Fields Using Illuminated Stream Lines." *Proceedings of the Visualization '96 Conference*, pp. 107–113. Oct 1996

[8] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. *Numerical Recipies in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, second edn., 1992

[9] Gumhold S., Straßer W. "Real Time Compression of Triangle Mesh Connectivity." M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pp. 133–140. ACM SIGGRAPH, Addison Wesley, Jul. 1998. ISBN 0-89791-999-8

[10] Rossignac J. "Edgebreaker: Connectivity compression for triangle meshes." Technical Report GIT-GVU-98-35, Georgia Institute of Technology, Oct. 1998

[11] Gotsman C., Gumhold S., Kobbelt L. "Tutorials on Multiresolution in Geometric Modelling." pp. 319–362, 2002

[12] Isenburg M., Snoeyink J. "Face Fixer: Compressing Polygon Meshes with Properties." *SIGGRAPH'00 Conference Proceedings*, pp. 263–270. 2000

[13] Alliez P., Desbrun M. "Valence-Driven Connectivity Encoding for 3D Meshes." *Eurographics'01 Conference Proceedings*, pp. 480–489. 2001

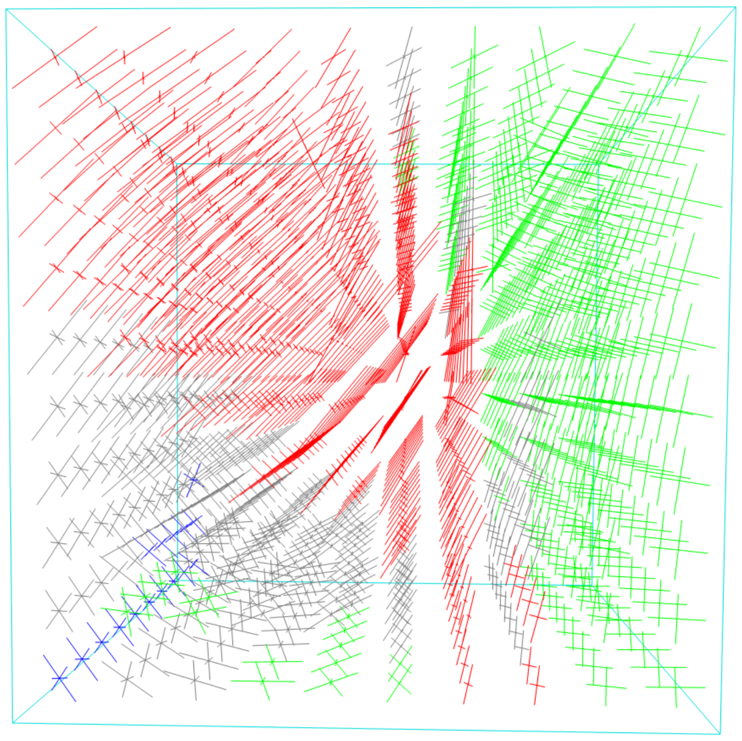

Figure 8: The preview for the dataset shown in Figure 9. Green is planar anisotropy, red linear. Shown are the axes of the eigenvectors.

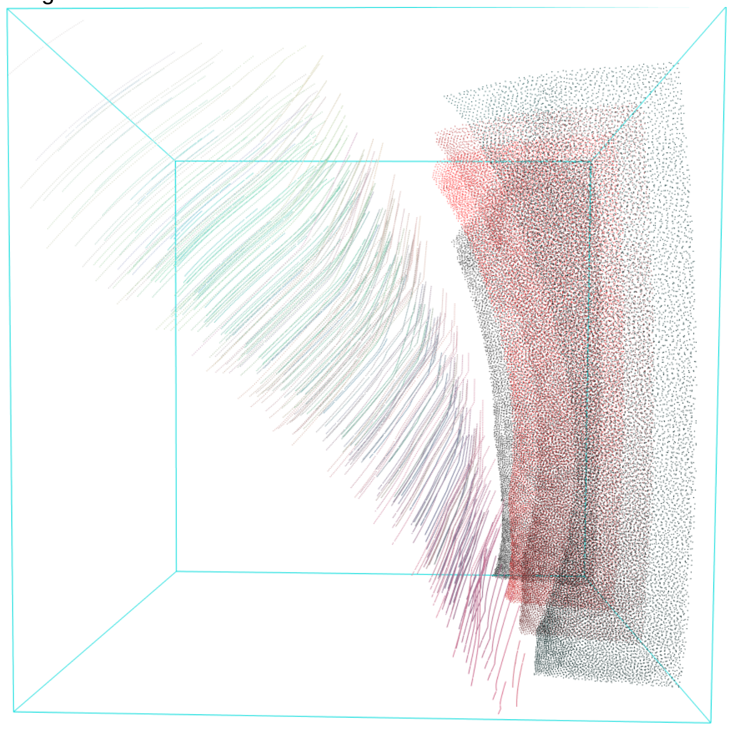

Figure 9: Stream tubes and diffusion surfaces rendered as point clouds.

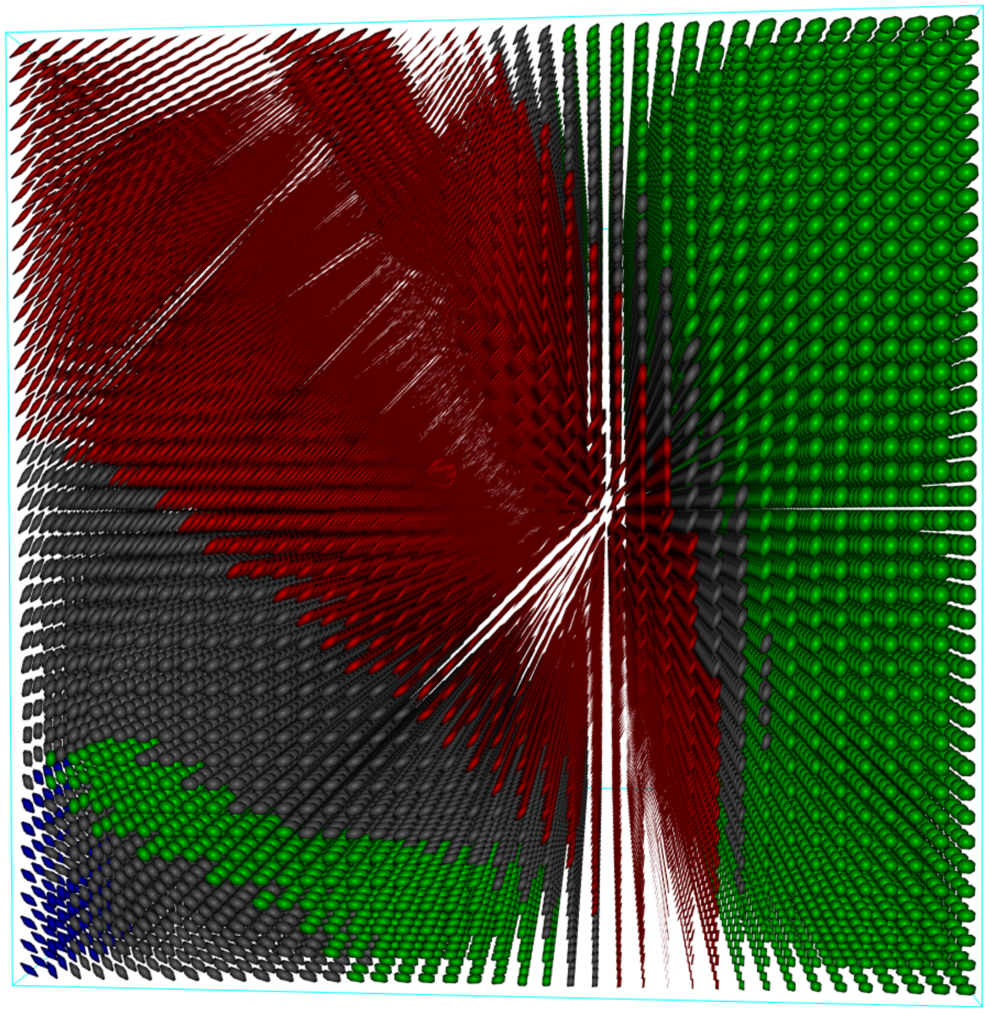

Figure 10: A classic approach for visualizing tensor fields is to render ellipsoids. Note the lack of visibility due to occlusion of the ellipsoids.

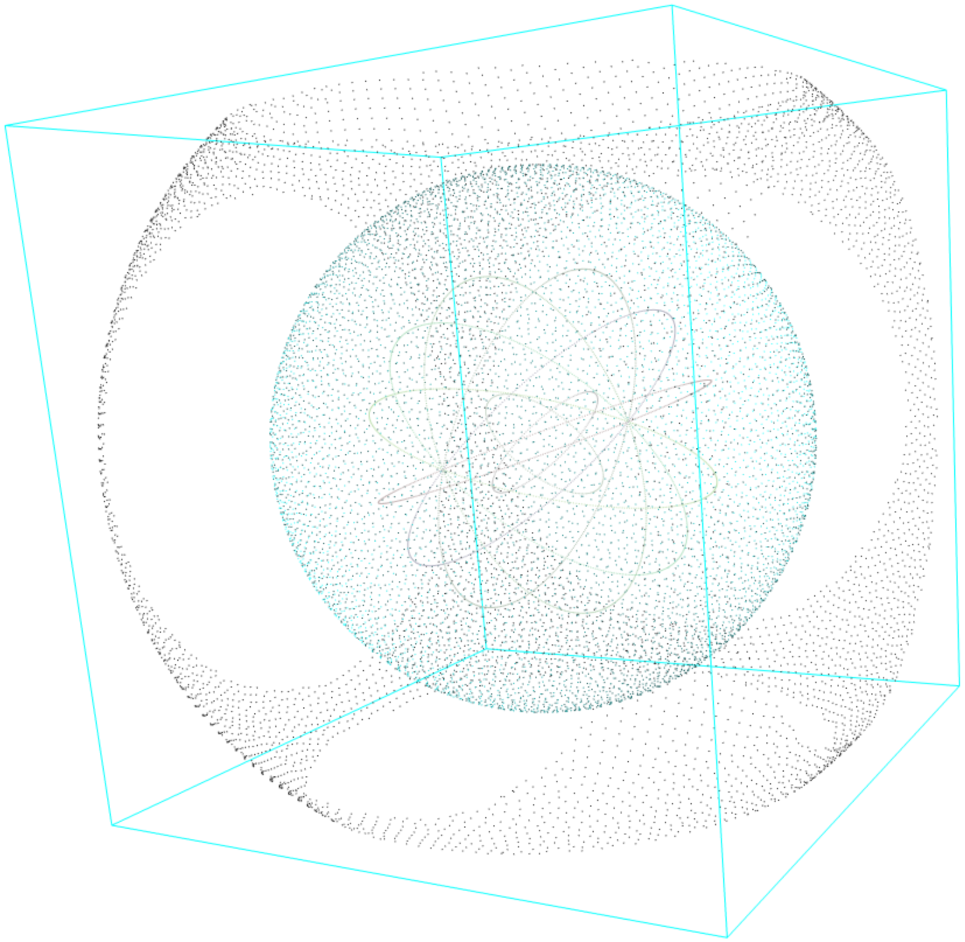

Figure 11: The power of point rendering. The interior stream lines are still visible although they are encapsulated by a closed surface.