Fabric-like Visualization of Tensor Field Data on Arbitrary Surfaces in Image Space

Sebastian Eichelbaum, Mario Hlawitschka, Bernd Hamann, and Gerik Scheuermann

Abstract Tensors are of great interest to many applications in engineering and in medical imaging, but a proper analysis and visualization remains challenging. It already has been shown that, by employing the metaphor of a fabric structure, tensor data can be visualized precisely on surfaces where the two eigendirections in the plane are illustrated as thread-like structures. This leads to a continuous visualization of most salient features of the tensor data set.

We introduce a novel approach to compute such a visualization from tensor field data that is motivated by image space line integral convolution (LIC). Although our approach can be applied to arbitrary, non-self-intersecting surfaces, the main focus lies on special surfaces following important features, such as surfaces aligned to the neural pathways in the human brain. By adding a postprocessing step, we are able to enhance the visual quality of the results, which improves perception of the major patterns.

1 Motivation and Related Work

Since the introduction of tensor lines and hyperstreamlines [6], there have been many research efforts directed at the continuous representation of tensor fields, including research on tensor field topology [11, 23, 22]. Zheng and Pang introduced HyperLIC [31], which makes it possible to display a single eigendirection of a tensor field in a continuous manner by smoothing a noise texture along integral lines, while neglecting secondary directions. Recent approaches to visualize Lagrangian struc-

Sebastian Eichelbaum · Gerik Scheuermann
Abteilung für Bild- und Signalverarbeitung, Institut für Informatik, Universität Leipzig, Germany, e-mail: {eichelbaum | scheuermann}@informatik.uni-leipzig.de

Mario Hlawitschka · Bernd Hamann
Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, CA, e-mail: {hlawitschka | bhamann}@ucdavis.edu

tures on tensor fields [12] provide information on one chosen tensor direction and are especially useful for diffusion tensor data, where the main tensor direction can be correlated to neural fibers or muscular structures, whereas the secondary direction only plays a minor role. More recently, Dick et al. [7] published an interactive approach to visualize a volumetric tensor field for implant planning.

Hotz et al. [13] introduced Physically Based Methods (PBM) for tensor field visualization in 2004 as a means to visualize stress and strain tensors arising in geomechanics. A positive-definite metric that has the same topological structure as the tensor field is defined and visualized using a texture-based approach resembling LIC [4]. Besides other information, eigenvalues of the metric can be encoded by free parameters of the texture definition, such as the remaining color space. Whereas the method's implementation for parameterizable surfaces that are topologically equivalent to discs or spheres is straightforward, implementations for arbitrary surfaces remains computationally challenging. In 2009, Hotz et al. [14] enhanced their approach to isosurfaces in three-dimensional tensor fields. A three-dimensional noise texture is computed in the data set and a convolution is performed along integral lines tangential to the eigenvector field. LIC has been used in vector field visualization methods to imitate *Schlieren* patterns on surfaces that are generated in experiments where a thin film of oil is applied to surfaces, which show patterns caused by the air flow. In vector field visualization, image space LIC is a method to compute *Schlieren*-like textures in image space [27, 28, 17, 9], intended for large and non-parameterized geometries. Besides the non-trivial application of image space LIC to tensor data, image space LIC has certain other drawbacks. Mainly because the noise pattern is defined in image space, it does not follow the movement of the surface and, therefore, during user interaction, the three-dimensional impression is lost. A simple method proposed to circumvent this problem is animating the texture pattern by applying randomized trigonometric functions to the input noise. Weiskopf and Ertl [26] solved this problem for vector field visualization by generating a three-dimensional texture that is scaled appropriately in physical space.

In contrast to other real-time tensor-field visualizations like [30], we developed and implemented an algorithm similar to the original PBM but for arbitrary non-intersecting surfaces in image space. Our algorithm can perform at interactive frame rates for large data sets on current desktop PCs. We overcome the drawbacks present in image space LIC implementations by defining a fixed parameterization on the surface. Thus, we do not require a three-dimensional noise texture representation defined at sub-voxel resolution for the data set. Our approach is capable of maintaining local coherence of the texture pattern between frames when (1) transforming, rotating, or scaling the visualization, (2) changing the surface by, e.g., changing isovalues or sweeping the surface through space, and (3) changing the level of detail. In addition, we implemented special application-dependent modes to ensure our method integrates well with existing techniques. Besides this, we also apply several postprocessing steps to further increase the visual quality and clarity of the shown structures.

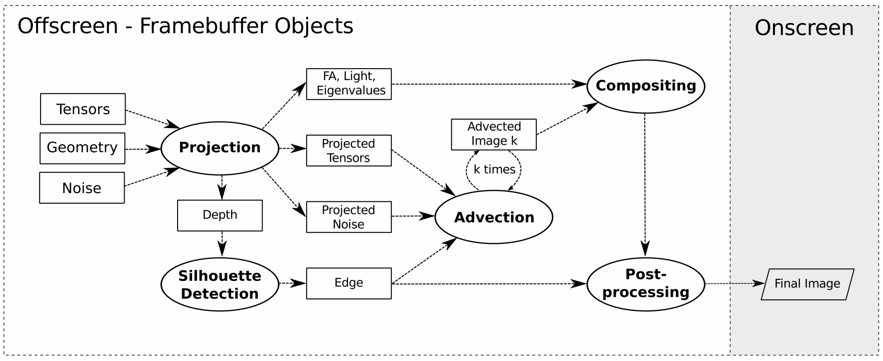

Fig. 1 Flowchart indicating the four major steps of the algorithm: **projection**, which transforms the data set in an image space representation and produces the initial noise texture on the geometry; **silhouette detection**, required for the advection step and the final rendering; **advection**, which produces the two eigenvector textures; **compositing**, which combines intermediate textures; and the **postprocessing**, which adds additional shading and improves the perceptual quality of the final visualization. Between consecutive steps, the data is transferred using textures.

2 Method

We employ a multi-pass rendering technique that consists of five major rendering passes as outlined in Figure 1. The complete pipeline is processed in one single render-frame and offscreen. In the following sections we describe each single step in our pipeline and imply that all operations are done on a per-pixel basis, if not denoted differently. We additionally rely on Figure 1 in many sections, as this figure shows all needed inputs and generated outputs of each step.

After generating the basic input textures once, the first pass calculates and projects all required data into image space. This encompasses the eigenvector-decomposition and projection, the noise projection to the surface, lighting and further color-mapping. Using the calculated eigenvalues, the fractional anisotropy is calculated as well. It is used later for clipping and color-mapping. Pass two performs a silhouette detection on the depth-buffer if the rendered geometry. That is used to guarantee integrity of the advection image, computed by multiple iterations of pass three. Pass three uses the projected eigenvectors to smear the projected noise on the surface in k iterations along the eigenvectors. Eventually, pass four composes the intermediate textures in an image, which is then post-processed by step five and rendered on-screen.

2.1 Initial Noise Texture Generation

In contrast to standard LIC approaches, to achieve a proper visual representation of the data, high-frequency noise textures, such as white noise, are not suitable for the compositing of multiple textures. Therefore, we compute the initial noise texture using the reaction diffusion scheme first introduced by Turing [24]. It simulates the mixture of two reacting chemicals, which leads to larger but smooth "spots" that are randomly and almost uniquely distributed (cf. Figure 2, right). This can be pre-computed on the CPU once. The created texture can then be used for all consecutive frames. For the discrete case, the governing equations are:

$$\begin{aligned} \Delta a_{i,j} &= F(i,j) + D_a \cdot (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} - 4 \cdot a_{i,j}), \\ \Delta b_{i,j} &= G(i,j) + D_b \cdot (b_{i+1,j} + b_{i-1,j} + b_{i,j+1} + b_{i,j-1} - 4 \cdot b_{i,j}), \text{where} \\ F(i,j) &= s(16 - a_{i,j} \cdot b_{i,j}) \text{ and } G(i,j) = s(a_{i,j} \cdot b_{i,j} - b_{i,j} - \beta_{i,j}). \end{aligned} \tag{1}$$

Here, we assume continuous boundary conditions to obtain a seamless texture in both directions. The scalar s allows one to control the size of the spots where a smaller value of s leads to larger spots. The constants D_a and D_b are the diffusion constants of each chemical. We use $D_a = 0.125$ and $D_b = 0.031$ to create the input textures. We gained both constants empirically. They directly influence the shape and density of the created spots.

2.2 Projection Step

The first step of our algorithm is the projection pass. Its purpose is to project the geometry as well as the tensorial data to image space. Besides this, the initial noise texture created earlier is mapped to the surface. After the projection step, the tensors, the noise and the rendered geometry are available in image space and can be used by the consecutive passes. As the next steps are all operating in image space, we state that all consecutive steps are done on a per-pixel basis. The operations of the projection step are done on a per-fragment basis. Only the projection of the geometry is done vertex-wise.

2.2.1 Projection into Image Space

In the first step, we project the data into image space by rendering the surface using the default OpenGL rendering pipeline. Notably, the surface does not need to be represented by a surface mesh. Any other representation that provides proper depth and surface normal information works just as well (e.g., ray-casting methods for implicit surfaces, cf. Knoll et al. [16]). In the same rendering step, the tensor field is transformed from world space to object space, i.e., each tensor T, that is interpolated at the point on the surface from the surrounding two- or three-dimensional tensor field is projected onto the surface by

$$T' = P \cdot T \cdot P^T, \tag{2}$$

with a matrix P defined using the surface normal n as

$$P = \begin{pmatrix} 1-n_x^2 & -n_y n_x & -n_z n_x \\ -n_x n_y & 1-n_y^2 & -n_z n_y \\ -n_x n_z & -n_y n_z & 1-n_z^2 \end{pmatrix}. \quad (3)$$

The camera viewing system configuration and the available screen resolution imply a super- or sub-sampling of the data. We obtain an interpolated surface tensor in every pixel, which is decomposed into the eigenvector/eigenvalue representation using a method derived from the one presented by Hasan et al. [10], only using iteration-free math functions. This causes a tremendous acceleration on the GPU. With this method, we calculate the three real-valued, orthogonal eigenvectors $v_{\lambda_{1-3}}$ and the corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$. In our method, we are only using the first two eigenvectors, showing the two main directions. The eigenvectors, still defined in object space, are projected into image space using the same projection matrices M_M and M_P used for projecting the geometry to image space. These usually are the standard *modelview* and *projection* matrices OpenGL offers:

$$v'_{\lambda_i} = M_P \times M_M \times v_{\lambda_i}, \text{ with } (i \in 1,2). \quad (4)$$

After the projection, the two eigenvectors are not necessarily orthogonal anymore.

2.2.2 Noise Texture Transformation

Mapping the initial texture to the geometry is a difficult and application-dependent task. Even though there exist methods to parameterize a surface, they employ restrictions to the surface (such as being isomorphic to discs or spheres), require additional storage for texture atlases (cf. [19, 15]) and, in general, require additional and often time-consuming pre-processing.

Another solution, proposed by Turk et al. [25], calculates the reaction diffusion texture directly on the surface. A major disadvantage of this method is the computational complexity. Even though these approaches provide almost distortion-free texture representations, isosurfaces, for example, may consist of a large amount of unstructured primitives, which increases the pre-processing time tremendously.

Whereas previously published approaches for image space LIC either use parameterized surfaces to apply the initial noise pattern to the surface or use locally or globally defined three-dimensional textures [26], we define an implicit parameterization of the surface that provides an appropriate mapping of the noise texture to the surface.

For our purpose, a simple, yet fast and flexible mapping strategy is used. We implicitly split the world space in voxels of equal size. These voxels fill the bounding volume of the geometry but are never created explicitly. The seamless noise texture is mapped onto each side of each voxel exactly once (no tiling). This creates a seamless mapping of the noise onto the surface of any connected block of voxels. During rendering, each point p of the surface can then be classified to belong to one

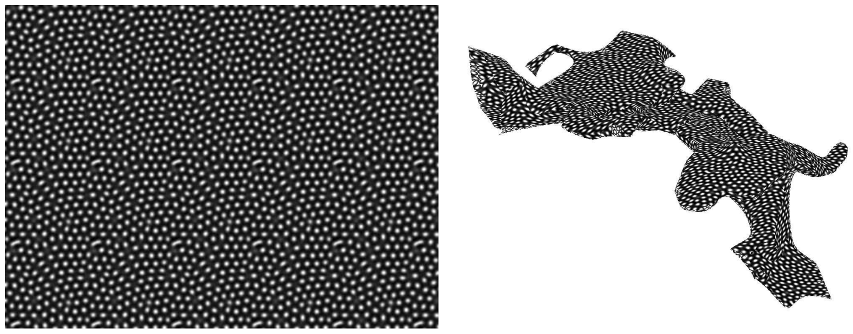

Fig. 2 Illustration of the reaction diffusion texture used (right) and the noise texture mapped to geometry (left).

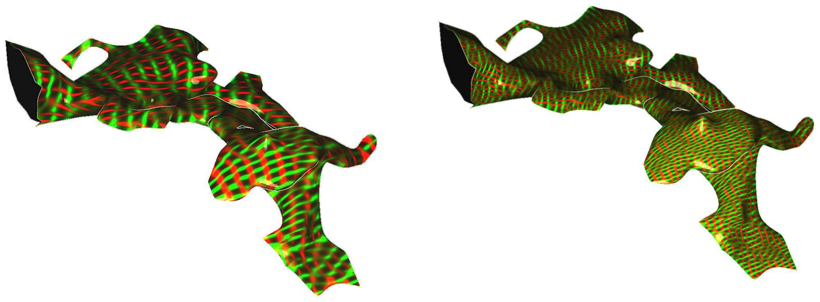

Fig. 3 Comparison of two different voxel sizes during noise mapping. This demonstrates the possibility for dynamic refinement of the input noise to achieve different levels of detail.

certain voxel. This can be interpreted as discretization of the surface with the help of the implicit voxels. The normal at p on the surface is then used to find the most similar side of the voxel associated with p. Therefore, the scalar product between the surface normal and the normals of each side are compared. Once the side-plane is found, the following table determines the point's p texture coordinates:

Side-normal	Texture coordinates
$(1,0,0)$ or $(-1,0,0)$	(p_y, p_z)
$(0,1,0)$ or $(0,-1,0)$	(p_x, p_z)
$(0,0,1)$ or $(0,0,-1)$	(p_x, p_y)

Please note, that we assume the texture coordinates to be defined in a wrapped and continuously defined coordinate system, which is common in OpenGL. This allows the seamless tiling of the input noise texture on each voxel surface, which then is mapped to the surface. This can be interpreted as an orthographic projection of the voxel side plane onto the surface along the plane's normal vector.

Regardless of its simplicity, this method supports a fast and flexible parameterization of the surface space that only introduces irrelevant distortions (cf. Figure 2), which vanish during the advection step.

By changing the size of voxels during the calculation, different frequencies of patterns can easily be produced and projected onto the geometry. This capability allows one to change the resolution of the texture as required for automatic texture refinement when zooming. A comparison of two different levels of detail is shown in Figure 3.

2.3 Silhouette Detection

Following the projection pass, the silhouette detection pass uses the rendered surface's depth as input. To avoid advection over geometric boundaries, a silhouette of the object is required to stop advection in these areas [17]. Otherwise, tensor advection would lead to a constant flow of "particles" across surface boundaries which makes the surface's geometry and topology unrecognizable.

A standard three-by-three Laplacian filter, defined by the convolution mask

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{5}$$

applied to the depth values followed by thresholding, has proven to be suitable for our purposes. The silhouette image e for each pixel (x,y) is then provided to the next pass.

2.4 Advection

We have discussed how to project the geometry and the corresponding tensor field to image space. With the prepared image space eigenvectors and the input noise texture on the geometry, the advection can be done. Another important input is the advected image of the previous advection pass, created during the last render-frame. For the first frame, the geometry mapped noise is used as initialization.

In the advection step, an Euler integration is applied to both vector fields separately. In our case, we do not calculate streamlines at each position of both vector fields, as normally done in LIC. We directly advect the noise input texture with the given vector fields, which provides the same results as locally filtering the data along pre-computed streamlines. During the advection pass, the previous advection results are, again, advected along the both eigenvector-fields separately. Each pass thereby only does one step along the two vector-fields. This decision was based on the fact that massively parallel architectures like modern GPUs are able to perform this task in parallel for each pixel several hundred times per second.

An important abortion-criteria here is the silhouette image e. If an edge is crossed, integration is stopped. The advection iteration can also be stopped if the

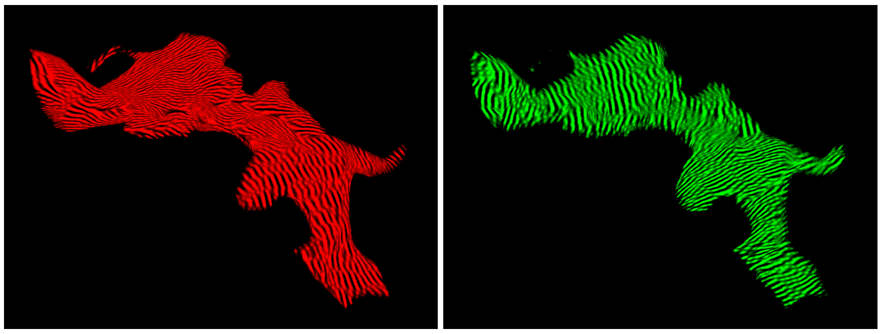

Fig. 4 Advection texture after ten iterations. Left: red channel containing advected noise along the eigenvectors v'_{λ_1}; Right: green channel containing the advected noise along the second eigenvectors v'_{λ_2}.

advection reaches a saturation; i.e. the resulting advected images do not differ from the previous ones. Due to this saturation effect, we do not use the iteration count k as abortion criterion. Since the eigenvectors do not have an orientation, the advection needs to be done in direction of v'_{λ_1} and $-v'_{\lambda_1}$ and for v'_{λ_2} and $-v'_{\lambda_2}$ respectively. At this point, we have two advected images for each eigenvector. These get composited equally to for both eigenvectors at this pixel. Furthermore, the advection results for each eigenvector again get blended with the input noise. The blending ratio between noise and the advected images determines how crisp the results are. Lower ratios produce crispier images. Higher ratios produce more smooth and more smeared images. Throughout this paper, we use a ratio of $\frac{1}{10}$.

The resulting images, one for each eigenvector, are then used as input during the next render-frame. They can be stored in one single texture in different color channels. We use the red color channel for the first eigenvector and the green color channel for the second one. Figure 4 shows the resulting images of the advection step after ten iterations ($k = 10$). For later reference, we denote these advected images after k steps with $A^k_{\lambda_1}$ and $A^k_{\lambda_2}$. The number of iterations k hereby equals the number of rendered frames as we do only one advection step per frame.

2.5 Compositing

In a subsequent rendering pass, an initial fabric-like texture is composed. For the sake of simplicity and the limitations of some graphics boards, we split the final image creation step in an initial compositing followed by a postprocessing step described in the next section. The compositing step combines the advection results $A^k_{\lambda_1}$ and $A^k_{\lambda_2}$ into one image, whereas the postprocessor mainly improves visual quality. The input of the compositing step are the both advected images $A^k_{\lambda_1}$ and $A^k_{\lambda_2}$, the depth-buffer, the silhouette e as well as the light and colormap information from the projection pass. On the GPU, these inputs get composed to the final RGB triple for each pixel:

$$\begin{aligned} R &= \frac{r \cdot A^k_{\lambda_2}}{8 \cdot (A^k_{\lambda_1})^2} + e + light, \\ G &= \frac{(1-r) \cdot A^k_{\lambda_1}}{8 \cdot (A^k_{\lambda_2})^2} + e + light, \\ B &= e + light. \end{aligned} \tag{6}$$

Equation 6 is a weighting function between the two advected images for both eigen-

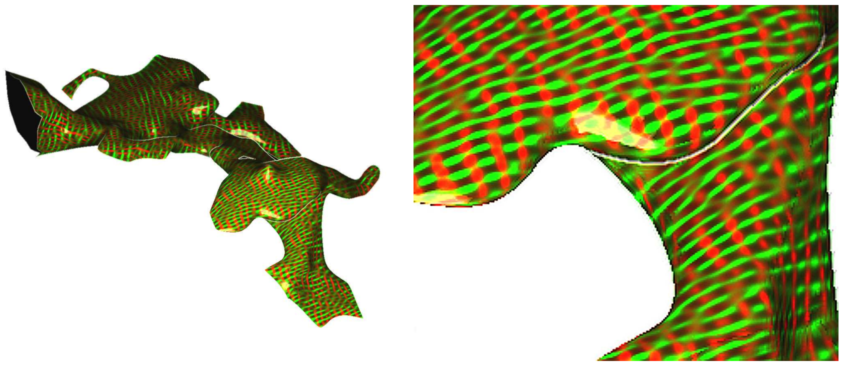

Fig. 5 The composited image produced by the compositing shader with lighting. Left: the whole geometry. Right: a zoomed part of the geometry to show the still blurry fabric structure on the surface.

vectors. The scalar factor r is used to blend between the two tensor directions. If both directions are equally important, a value of 0.5 ensures an equal blending of both directions. To explain the above compositing scheme, we are using the red component as an example. The red color should represent the main tensor direction. We therefore reduce the intensity of the second eigenvector image $A^k_{\lambda_2}$ using the overemphasized first eigenvector image $A^k_{\lambda_1}$. To furthermore emphasize the influence of a high intensity in the advected image for the first eigenvector, the denominator is squared. This way, pixels with a high intensity in the first eigenvector direction get a high red intensity. This is done vice versa for the green channel. The compositing implicitly utilizes the clamping to $[0,1]$ which is done for colors on the GPU. This approach creates a mesh resembling the tensor field's structure. To reduce the effect of light sources on the color coding, we use a separate lighting function *light* that, while manipulating the intensity, does not affect the base color of the mesh structure. The geometry's shape is furthermore emphasized using the silhouette image e. Even though Blinn-Phong shading [2] provides the required depth cues, additional emphasis of the third dimension using depth-enhancing color coding has proven to provide a better overall understanding of the data [5]. These techniques can be incorporated in our compositing scheme easily.

2.6 Postprocessing

Additional filters can be applied to the composed image, such as contrast enhancement or sharpening filters, which are commonly used in vector field LIC [26, 9]. Figure 5 shows the result of Equation 6 combined with Blinn-Phong shading. The results still look blurry and justify the need for additional postprocessing.

Bump mapping, first introduced by Blinn [3] to simulate three-dimensionality in planar surfaces, can be used to improve spatial perception of the fabric surface. As bump mapping is normally computed in world space, where the three-dimensional tangent space is known, the textured surface would be required in world space, whereas our texture is parameterized for use in image space. Transforming the modified normal, which is required for bump mapping and, in fact, depends on the gradient information on the surface, from image space back to world-space is not a trivial task, especially when using a perspective projection. Therefore, we use a modified approach that can be applied in image space only.

Bump mapping requires the surface normal at each point (x,y) of the surface, which can be obtained using the gradient information on each pixel of the surface in image space:

$$g(x,y) = ||\nabla(R+G)(x,y)||. \tag{7}$$

The resulting two-dimensional vector $g(x,y)$ describes the gradient on the image plane using each pixel's intensity. The blue color channel is not used as it does not contain relevant information besides lighting and edges. It is also worth noting that we exclude the light $\mathscr{L}_{x,y}$ and edge information $e_{x,y}$ from gradient calculation, as we do lighting using bump mapping. Using this gradient, the new surface normal is a weighted sum of the surface normal and the gradient, and is used for calculating Phong lighting $\mathscr{B}_{x,y}$ as seen in Figure 6.

Figure 6 (right) shows the additional scaling of the red and green color channels by the original color intensities, to lead to a more fabric-like impression of the lines. Equation 8 shows this in more detail:

$$\begin{aligned} R_b(x,y) &= \mathscr{B}_{x,y}(R(x,y)G(x,y)+R^2(x,y))+e_{x,y}+light(\mathscr{L}_{x,y}), \\ G_b(x,y) &= \mathscr{B}_{x,y}(R(x,y)G(x,y)+G^2(x,y))+e_{x,y}+light(\mathscr{L}_{x,y}). \end{aligned} \tag{8}$$

With the help of bump mapping, we achieve a better spatial impression of the fabric-like pattern. Besides this, postprocessing filters can help to avoid blurry structures.

A further visual improvement can be achieved by interpreting the structure on the surface as streamtubes [29] along the surface. Therefore an approach similar to the ones in [18, 20] is appropriate to create the visual effect of streamtubes on the

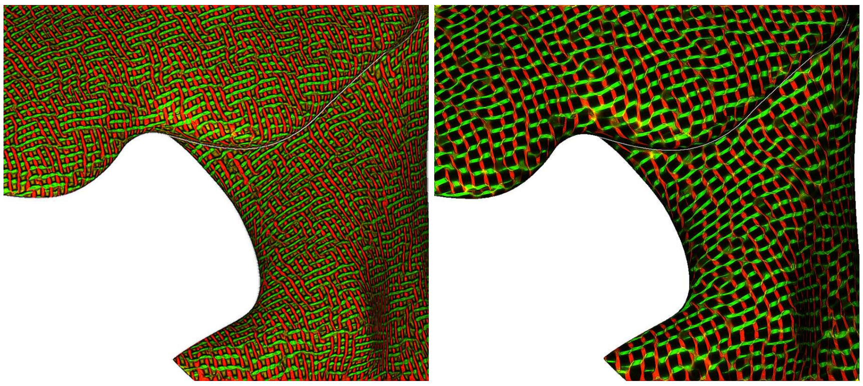

Fig. 6 The final image produced by the postprocessing shader in combination with bump mapping, the geometry's Phong shading and combined edges. Left: standard bump mapping. Right: the same zoomed part of the original geometry to show the effect of weighting the resulting Phong intensities by the original $R(x,y)$ and $G(x,y)$ intensities. This approach creates a more fabric-like impression that can be misunderstood as rotating ribbons similar to stream ribbons.

geometry's surface, without actually creating tubes. First, we need to have some kind of tangential coordinate system, similar to the one needed for bump mapping. Since our bump mapping is done in image space, the normal of the image plane is $(0,0,1)^T$. The eigenvectors v'_{λ_1} and v'_{λ_2} in Equation 4 from the tensor field in image space can be used as the tangent for each field. These tangents denote the direction of the tube along the surface and, together with the normal, define the binormal vector, which is nearly equal to the gradient vector. In practice, it normally is not exactly the same. The binormal b for each eigenvector field i is defined as:

$$b_i = (0,0,1)^T \times v'_{\lambda_i}, \text{ with } i \in \{0,1\}. \tag{9}$$

These binormals can be calculated for each point on the surface. To finally determine the point's actual position on the tube, described by the eigenvectors v'_{λ_i}, one has to find the border of the fabric structure that has been created by the compositing step. Mathematically, this can be expressed in this way:

$$\begin{aligned} B =& \{s|R(sb_1) < \varepsilon \wedge s \in \mathbb{R}\} \wedge (a_p, a_n) \in \mathscr{P}(B), \text{ with} \\ a_p =& min\{s|s \in B \wedge s \geq 0\} \\ a_n =& max\{s|s \in B \wedge s < 0\}. \end{aligned} \tag{10}$$

In other words, we find the smallest scaling factors a_n and a_p which scale the binormal vectors b_1 and b_2 in both directions, so that they point on a area below a given threshold ε in the composited image from the prior step, therefore pointing to the border of the tube. As the mapping functions R and G, from Equation 6, only need two-dimensional positions x and y, the binormal's x and y-components are used and the z-component is ignored, as it is always zero. The same factors a_p and a_n for the second eigenvector field are calculated using the green color channel of the composited image, which are used in the same way as described below to render the tubes for the second eigenvector field. The width of the tube at a given point is defined by $a_p + a_n$. The width of the tube is set in relation with the factor a_p to find the actual position of the current point (x,y) on the tube by using:

$$p = \frac{a_p}{a_p + a_n} \in [0,1]\,, \tag{11}$$

which finally is squared to describe the round surface of a tube:

$$r = \begin{cases} (1-2p)^2 & \text{if } p \geq 0 \\ -(1-2p)^2 & \text{if } p < 0 \end{cases} \text{ with } ratio \in [-1,1]\,. \tag{12}$$

The value of r describes the ratio between the normal completely on the plane (with a zero z-component) and the normal completely pointing towards the camera (with a z-component of one):

$$n = (1-r)(0,0,1)^T + rb_0 \tag{13}$$

The normal n is used to calculate the Phong shading on the surface and produces the tube-like effect with proper spatial impression on the surface, as can be seen in Figure 7.

The artifacts seen in Figure 7 result from the local approach we are using to calculate the tubes. As we do not integrate along the eigenvector-field, there may be discontinuities along a tube in the produced image. There are also artifacts caused by a blurry input field, where borders cannot be found clearly. But, since the frequency of the fabric structure is normally much higher, these effects are not visible anymore, as can be seen in Figure 7, left.

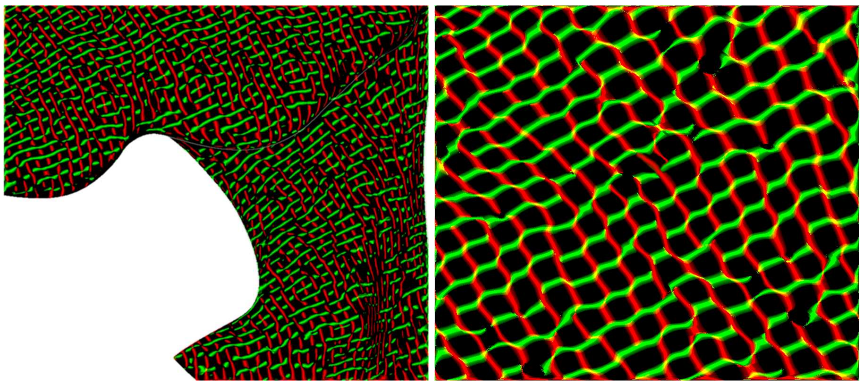

Fig. 7 Left: Interpreting the final image from Figure 5 as streamtubes along the geometry's surface, and lighting them accordingly, results in a less blurry surface. Right: zoomed part of the left geometry to show the tube effect. Although there are plenty of artifacts in the zoomed image, they do not influence the overall impression of images not zoomed as much. Especially, such strongly zoomed images are not useful for gathering an overview over the tensor field's structure.

2.7 Implementation

The implementation of the pipeline shown in Figure 1 is straight forward. The figure clearly shows the input and output textures of each step and their execution order. The whole pipeline is implemented using OpenGL and framebuffer objects (FBO), which allow the efficient offscreen rendering and image space based processing we need. The projection step is the beginning of the pipeline and the only step which is not in image space. For the consecutive steps, we render a quad, filling the whole viewport of the FBO. The inputs and outputs are then bound as textures to the FBO and the quad respectively. Since texture space is limited on the hardware, it is important to store as much information as possible in each texture (four channels per texture available). The steps itself are all implemented as fragment shaders using GLSL. This way, we can work on a per-pixel basis easily. There are only some implementation specifics we want to mention here.

Projection Step

Our implementation is not limited to a special kind of geometry. It is able to handle almost every tensor field defined on a surface. It is, for example, possible to calculate an isosurface on a derived scalar metric, like fractional anisotropy, or on a second data set to generate a surface in a three-dimensional data domain. Other methods include hyper-stream surfaces [6], wrapped streamlines [8], or domain-dependent methods like dissection-like surfaces presented in [1]. The only requirement for the surface is that it is non-self-intersecting and that smooth normals are provided as they are required for the projection step and for proper lighting.

As the tensors are symmetric, it is sufficient to transfer six floating-point values per vertex to the GPU. In our case, two three-dimensional texture coordinates are used per vertex to upload the tensor information along with the geometry. Assuming the tensor T is available on the GPU, it is possible to map the two main directions to the surface described by the normal n at the current vertex using Equation 2. This projection is implemented in a per-vertex manner in the vertex shader. In contrast, to ensure proper interpolation, eigenvalue decomposition and eigenvector calculation together with image space projection need to be done in the fragment shader. Since the eigenvectors are without orientation, it is possible to have sign flips between adjacent vertices. If the interpolation takes place after the eigenvector decomposition, these sign changes can render the interpolation useless. The eigenvectors v'_{λ_1} and v'_{λ_2} need to be scaled since textures are used for transportation where each value must be in the interval $[0,1]$. To simplify further data handling and storage on the GPU, we scale the eigenvectors as follows:

$$\|v\|_\infty = max\{|v_x|, |v_y|\} \tag{14}$$

$$v''_{\lambda_i} = \frac{v'_{\lambda_i}}{\|v'_{\lambda_i}\|_\infty} \text{ with } i \in \{1,2\}, \quad \text{and} \quad \|v'_{\lambda_i}\|_\infty \neq 0 \tag{15}$$

The maximum norm (L_∞-norm) ensures that one component of the eigenvector is 1 or -1 and, therefore, one avoids numerical instabilities arising when limited storage precision is available, and can use memory-efficient eight-bit textures. The special case $\|v'_{\lambda_i}\|_\infty = 0$ only appears when the surface normal and the eigenvector point in the same direction. This case needs to be handled in the shader.

Advection Step

During each advection iteration, the input and output texture need to be switched. This way, the advection result of the previous advection iteration can be used as input without the need to allocate and deallocate a separate texture for each iteration.

3 Results

We have introduced a method to create a fabric-like surface tensor LIC in image space, similar to the one introduced in [13]. We used ideas from [17] to transform the algorithm into image space. Our implementation, using this method, is able to reach frame rates high enough for real-time user interaction. The only bottleneck is the hardware's ability to render large and triangle-rich geometry. All further steps can be done in constant time, see Table 1.

3.1 Artificial Test Data Sets

We first applied our method to artificial test data sets that have complex topology: a torus, the Bretzel5, and the Tangle data set (cf. [16]), defined as implicit surfaces:

$$(1-\sqrt{x^2+y^2})(1-\sqrt{x^2+y^2})+z^2-0.125=0, \tag{16}$$

$$((x^2+.25*y^2-1)*(.25*x^2+y^2-1))^2+z^2-0.1=0, \text{ and} \tag{17}$$

$$x^4-5*x^2+y^4-5*y^2+z^4-5*z^2+11.8+w=0. \tag{18}$$

We used the Laplacian on the surfaces as tensor fields. The results displayed in Figure 8 show that neither the topology nor our artificial parameterization of the input noise texture influences the quality of the final rendering.

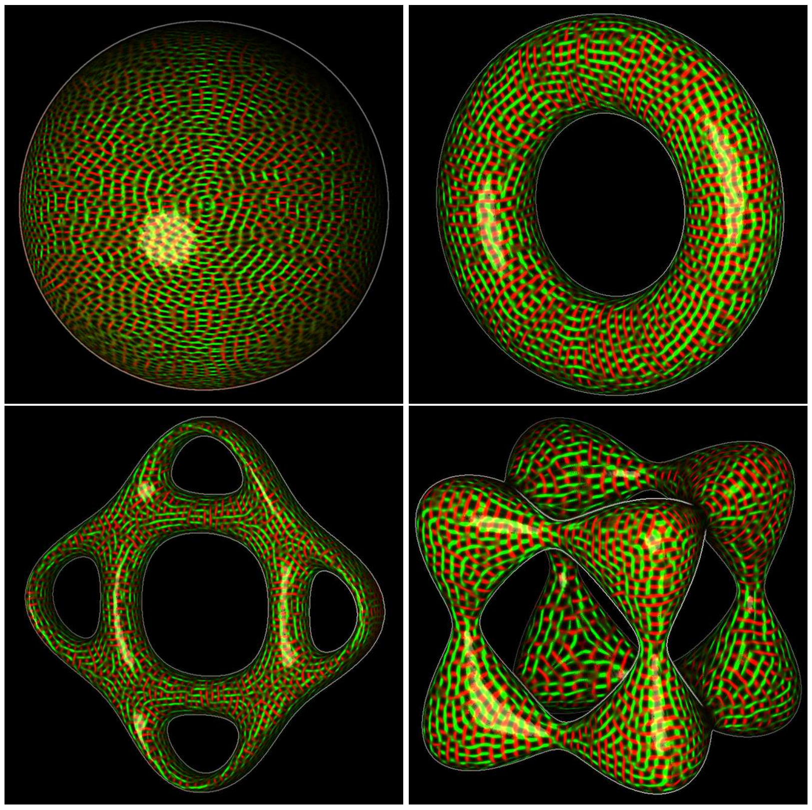

Fig. 8 Analytic test data sets. We applied our method to isosurfaces and the scalar field's Laplacian to demonstrate the suitability for complicated surfaces. Shown are the final images using our method for a sphere, torus, Tangle, and Bretzel5 data set (Equations 16–18).

3.2 Modification for Medical Data Processing

Even though many higher-order methods have been proposed, due to scanner, time, and cost limitations, second-order tensor data is still dominant in clinical application. Medical second-order diffusion tensor data sets differ from engineering data sets because they indicate one major direction whereas the secondary and ternary directions only provide information in areas where the major direction is not well-defined, i.e., the fractional anisotropy—a measure for the tensor shape—is low. Almost spherical tensors, which indicate isotropic diffusion, occur in areas where multiple fiber bundles traverse a single voxel of the measurement or when no directional structures are present. Therefore, we modulate the color coding using additional information: In areas where one fiber direction dominates, we only display this major direction using the standard color coding for medical data sets, where x, y, and z alignment are displayed in red, green, and blue, respectively. In areas where a secondary direction in the plane exists, we display this information as well but omit the

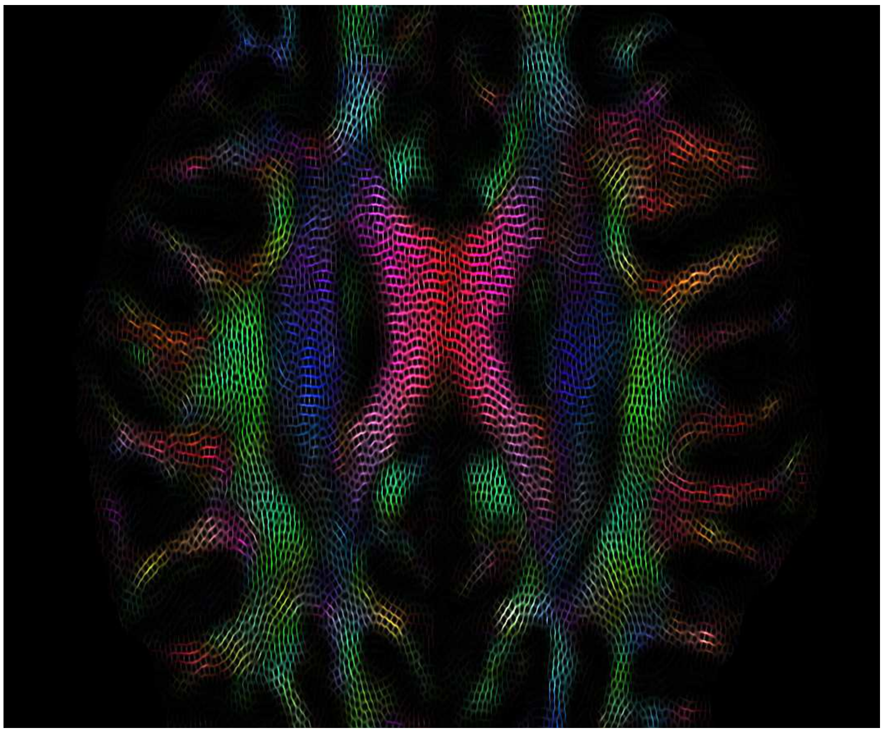

Fig. 9 An axial slice through a human brain: Corpus callosum (CC) (red), pyramidal tract (blue), and parts of the cinguli (green in front and behind the CC) are visible. The main direction in three-dimensional space is indicated by the RGB color map, where red indicates lateral (left–right), green anterior–posterior, and blue superior–inferior direction. The left–right structure of the CC can clearly be seen in its center, whereas color and pattern indicate uncertainty towards the outer parts. The same is true for the cinguli's anterior–posterior structure. As seen from the blue color, the pyramidal tract is almost perpendicular to the chosen plane and, therefore, secondary and ternary eigenvectors dominate the visualization. Alternatively, we could easily fade out those out-of-plane structures in cases where they distract the user.

secondary color coding and display the secondary direction in gray-scale rendering mode and always below the primary direction (cf. Figure 10). We use the method of Anwander et al. [1] to extract surfaces that are, where possible, tangential to the fiber directions. Hence, we can guarantee that the projection error introduced by our method in the surface's domain remains small. Even in areas where the fractional anisotropy is low and the color coding does no longer provide directional information, such as in some parts of the pyramidal tract in Figure 10, the texture pattern still provides this information.

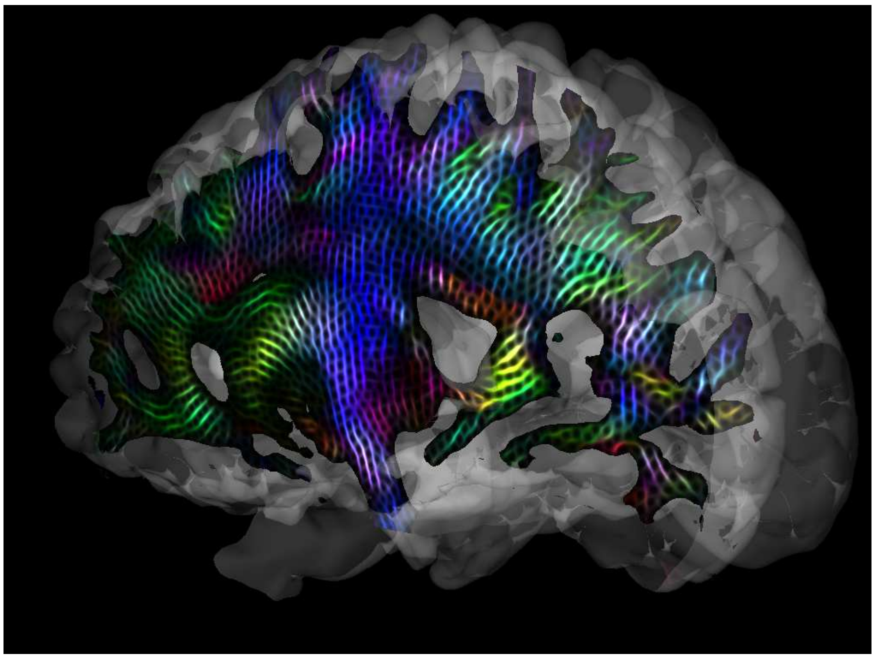

Fig. 10 Diffusion tensor data set of a human brain. We employed the method by Anwander et al. [1] to extract a surface following neural fibers and applied our method with an alternative color coding that is more suitable and can be incorporated more easily into medical visualization tools.

3.3 Mechanical Datasets

Our approach is not only applicable to medical datasets, but it can also be applied to many other tensor data sets. Figures 11 and 12 show a slice in an earthquake dataset and an analytical strain tensor field. The analytical data set is the well-known single point load data set, where a single infinitesimally small point source pushes on an infinite surface. The forces and distortions inside the object are represented by stress and strain tensors, which are symmetric, second-order tensors. The earthquake data set is a simulation of a single concrete pile in solid ground excited by a measured earthquake pattern from the Kyoto earthquake (cf. Figure 12). As shown, the material stress tensors, are defined on an irregular grid. We extracted a plane perpendicular to the pile and show the tensor information in that plane. Due to the time-dependent nature of the simulation, static images are quire complex.

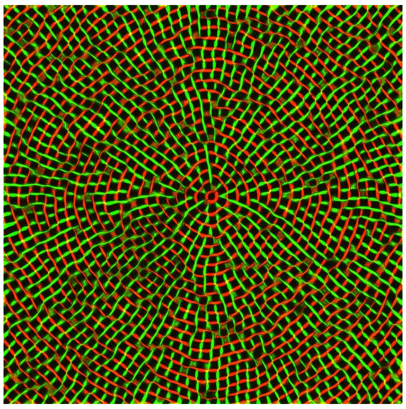

Fig. 11 A slice in the well-known single point load data set, showing the symmetric strain tensor at the surface of the slice.

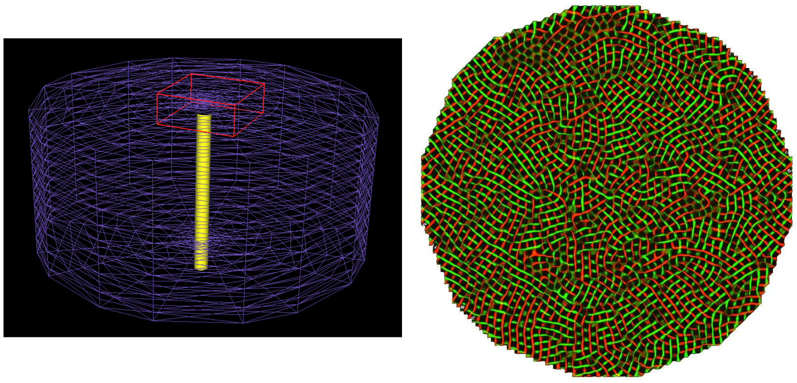

Fig. 12 A concrete pile in solid ground. Left: the original grid shown in purple. Right: a slice of the dataset showing the symmetric part of the tensor field.

3.4 Performance

As indicated before, the only "bottleneck" in the visualization pipeline is the strongly geometry-dependent projection step. Since the surface needs to be rendered repeatedly in case of user interaction, the performance measures of our method consider repeated rendering of the geometry. The frame rate with geometry not being moved and, therefore, making the projection step and the edge detection step unnecessary, is considerably higher. The advection step can be done multiple times per frame. This reduces the number of frames needed until the advection is saturated. To ensure high frame-rates and smooth user-interaction, we do only one advection step per frame. To make the frame rates in the following tables comparable, user interaction is assumed and, therefore, rendering a single frame always consists of

- one projection step, including geometry rendering;
- one edge detection pass;
- three advection iterations; and
- one output processing pass.

As seen in the previous sections, fragments not belonging to the geometry are discarded as soon as possible without using deferred shading. This also leads to performance gain in advection and output processing. In Table 1, a selection of data sets with their corresponding number of triangles and tensors are listed. The frame rates shown were obtained on an AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ (512K L2 Cache) with a NVIDIA G80 GPU (GeForce 8800 GTS) and 640MB of graphics memory at a resolution of 1024×768 pixels.

Figure	Nb Triangles	Nb Tensors	fps	fps (Phong only)	∅ Geometry Share
10	41472	63075	32	61	72%
5	58624	88803	30	60	69%
9	571776	861981	14	16	90%

Table 1 Frames per second (fps) for different data sets with given number of triangles and numbers of tensors. The frame rates are compared to simple rendering of the geometry using Phong shading. The frame rates were obtained for an AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ (512K L2 Cache) with an NVIDIA G80 GPU (GeForce 8800 GTS) and 640MB of graphics memory at a resolution of 1024×768 pixels. The geometry share relates the time used by the GPU to rasterize the geometry to the overall rendering time, which contains all steps of the pipeline. The time used to render the geometry clearly dominates the rendering times and reaches up to 90 percent of the overall rendering time even for medium-sized geometries.

The assumption that geometry rendering with projection is the weakest component in this pipeline and that edge detection, advection, and output processing perform at a data-independent frame rate is confirmed by the frame rates shown in Table 1. It confirms that for large geometries, rendering the geometry alone is the dominant component. Since the vertex-wise calculations during projection are limited to tensor projection (Equation 2) and noise texture transformation (Section 2.2.2), the most expensive calculations during projection are executed per fragment. This means that the expensive eigenvalue decomposition and eigenvector calculations are only required for fragments (pixels). To further decouple the calculation effort from the geometry's size, the depth test should be performed before performing the eigenvector decomposition. This goal can be achieved by first rendering the projected tensors to a texture, and computing the decomposition for visible fragments only. Nevertheless, this is not necessary for our current data set and screen sizes where the time required to render the geometry itself clearly dominates the time required to compute the texture pattern in image space. This can be seen in the increasing values in Table 1 with increasing size of vertices rendered.

4 Conclusions and Possible Directions for Future Research

We have presented a novel method for rendering fabric-like structures to visualize tensor fields on almost arbitrary surfaces without generating three-dimensional textures that span the whole data set at sub-voxel resolution. Therefore, our method can be applied to complex data sets without introducing texture memory problems common to methods relying on three-dimensional noise textures. As major parts of the calculation are performed in image space, the performance of our algorithm is almost independent of data set size, provided that surfaces can be drawn efficiently, e.g., by using acceleration structures to draw only those parts of the geometry that intersect the view frustum or using ray tracing methods.

Whether the surface itself is the domain of the data, a surface defined on the tensor information (e.g., hyperstream surfaces), or a surface defined by other unrelated quantities (e.g., given by material boundaries in engineering data or anatomical structures in medical data) is independent from our approach. Nevertheless, the surface has to be chosen appropriately because only in-plane information is visualized. To overcome this limitation, information perpendicular to the plane could be incorporated in the color coding, but due to a proper selection of the plane that is aligned with our features of interest, this has not been necessary for our purposes.

Especially in medical visualization, higher-order tensor information is becoming increasingly important and different methods exist to visualize these tensors, including local color coding, glyphs, and integral lines. Nevertheless, an extension of our approach is one of our major aims. In brain imaging, experts agree that the maximum number of possible fiber directions is limited. Typically, a maximum of three or four directions in a single voxel are assumed (cf. Schultz et al. [21]). Whereas the number of output textures can easily be adapted, the major remaining problem is a lack of suitable decomposition algorithms on the GPU. image space techniques, by their very nature, resample the data and, therefore, require one to use such proper interpolation schemes. In addition, maintaining orientations and assigning same fibers in higher-order data to the same texture globally is not possible today and, therefore, is a potential topic for further investigation.

Acknowledgements

We thank Alfred Anwander and Thomas R. Knösche from the Max Planck Institute for Human Cognitive and Brain Sciences, Leipzig, Germany, for providing the human brain image data sets, and for fruitful discussions and comments, and Boris Jeremić, Department of Civil and Environmental Engineering, UC Davis, for providing the earthquake data set. We thank the members of the Visualization and Computer Graphics Research Group of the Institute for Data Analysis and Visualization, Department of Computer Science, UC Davis, and the members of the Abteilung für Bild- und Signalverarbeitung des Instituts für Informatik der Universität Leipzig,

Germany.
This work has been supported by NSF grant CCF-0702817.

References

1. Anwander, A., Schurade, R., Hlawitschka, M., Scheuermann, G., Knösche, T.: White matter imaging with virtual klingler dissection. NeuroImage **47**(Supplement 1), S105 – S105 (2009). DOI 10.1016/S1053-8119(09)70916-4. Organization for Human Brain Mapping 2009 Annual Meeting
2. Blinn, J.F.: Models of light reflection for computer synthesized pictures. In: SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques, pp. 192–198. ACM, New York, NY, USA (1977). DOI http://doi.acm.org/10.1145/563858.563893
3. Blinn, J.F.: Simulation of wrinkled surfaces. SIGGRAPH Comput. Graph. **12**(3), 286–292 (1978). DOI http://doi.acm.org/10.1145/965139.507101
4. Cabral, B., Leedom, L.C.: Imaging vector fields using line integral convolution. In: SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp. 263–270. ACM, New York, NY, USA (1993). DOI http://doi.acm.org/10.1145/166117.166151
5. Chu, A., Chan, W.Y., Guo, J., Pang, W.M., Heng, P.A.: Perception-aware depth cueing for illustrative vascular visualization. In: BMEI '08: Proceedings of the 2008 International Conference on BioMedical Engineering and Informatics, pp. 341–346. IEEE Computer Society, Washington, DC, USA (2008). DOI http://dx.doi.org/10.1109/BMEI.2008.347
6. Delmarcelle, T., Hesselink, L.: Visualization of second order tensor fields and matrix data. In: VIS '92: Proceedings of the 3rd conference on Visualization '92, pp. 316–323. IEEE Computer Society Press, Los Alamitos, CA, USA (1992)
7. Dick, C., Georgii, J., Burgkart, R., Westermann, R.: Stress tensor field visualization for implant planning in orthopedics. IEEE Transactions on Visualization and Computer Graphics **15**(6), 1399–1406 (2009). DOI http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.184
8. Enders, F., Sauber, N., Merhof, D., Hastreiter, P., Nimsky, C., Stamminger, M.: Visualization of white matter tracts with wrapped streamlines. In: C.T. Silva, E. Gröller, H. Rushmeier (eds.) Proceedings of IEEE Visualization 2005, pp. 51–58. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos, CA, USA (2005)
9. Grabner, M., Laramee, R.S.: Image space advection on graphics hardware. In: SCCG '05: Proceedings of the 21st spring conference on Computer graphics, pp. 77–84. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1090122.1090136
10. Hasan, K.M., Basser, P.J., Parker, D.L., Alexander, A.L.: Analytical computation of the eigenvalues and eigenvectors in DT-MRI. Journal of Magnetic Resonance **152**(1), 41 – 47 (2001). DOI 10.1006/jmre.2001.2400
11. Hesselink, L., Levy, Y., Lavin, Y.: The topology of symmetric, second-order 3d tensor fields. IEEE Transactions on Visualization and Computer Graphics **3**(1), 1–11 (1997). DOI http://dx.doi.org/10.1109/2945.582332
12. Hlawitschka, M., Garth, C., Tricoche, X., Kindlmann, G., Scheuermann, G., Joy, K.I., Hamann, B.: Direct visualization of fiber information by coherence. International Journal of Computer Assisted Radiology and Surgery, CARS, CUARC.08 Special Issue (2009)
13. Hotz, I., Feng, L., Hagen, H., Hamann, B., Joy, K., Jeremic, B.: Physically based methods for tensor field visualization. In: VIS '04: Proceedings of the conference on Visualization '04, pp. 123–130. IEEE Computer Society, Washington, DC, USA (2004). DOI http://dx.doi.org/10.1109/VIS.2004.80
14. Hotz, I., Feng, Z.X., Hamann, B., Joy, K.I.: Tensor field visualization using a fabric-like texture on arbitrary two-dimensional surfaces. In: T. Möller, B. Hamann, R.D. Russel (eds.)

Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration. Springer-Verlag Heidelberg, Germany (2009)

15. Iwakiri, Y., Omori, Y., Kanko, T.: Practical texture mapping on free-form surfaces. In: PG ’00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications, p. 97. IEEE Computer Society, Washington, DC, USA (2000)
16. Knoll, A., Hijazi, Y., Hansen, C., Wald, I., Hagen, H.: Interactive ray tracing of arbitrary implicits with simd interval arithmetic. In: RT ’07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing, pp. 11–18. IEEE Computer Society, Washington, DC, USA (2007). DOI http://dx.doi.org/10.1109/RT.2007.4342585
17. Laramee, R.S., Jobard, B., Hauser, H.: Image space based visualization of unsteady flow on surfaces. In: VIS ’03: Proceedings of the 14th IEEE Visualization 2003 (VIS’03), p. 18. IEEE Computer Society, Washington, DC, USA (2003). DOI http://dx.doi.org/10.1109/VISUAL.2003.1250364
18. Merhof, D., Sonntag, M., Enders, F., Nimsky, C., Hastreiter, P., Greiner, G.: Hybrid visualization for white matter tracts using triangle strips and point sprites. IEEE Transactions on Visualization and Computer Graphics **12**(5), 1181–1188 (2006). DOI http://doi.ieeecomputersociety.org/10.1109/TVCG.2006.151
19. Purnomo, B., Cohen, J.D., Kumar, S.: Seamless texture atlases. In: SGP ’04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, pp. 65–74. ACM, New York, NY, USA (2004). DOI http://doi.acm.org/10.1145/1057432.1057441
20. Schirski, M., Kuhlen, T., Hopp, M., Adomeit, P., Pischinger, S., Bischof, C.: Virtual tubelets-efficiently visualizing large amounts of particle trajectories. Comput. Graph. **29**(1), 17–27 (2005). DOI http://dx.doi.org/10.1016/j.cag.2004.11.004
21. Schultz, T., Seidel, H.P.: Estimating crossing fibers: A tensor decomposition approach. IEEE Transactions on Visualization and Computer Graphics **14**(6), 1635–1642 (2008). DOI http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.128
22. Tricoche, X.: Vector and tensor field topology simplification, tracking, and visualization. Ph.D. thesis, University of Kaiserslautern, Germany (2002)
23. Tricoche, X., Scheuermann, G., Hagen, H.: Tensor topology tracking: A visualization method for time-dependent 2D symmetric tensor fields. In: Eurographics 2001 Proceedings, Computer Graphics Forum 20(3), pp. 461–470. The Eurographics Association, Saarbrücken, Germany (2001). DOI http://dx.doi.org/10.1111/1467-8659.00539
24. Turing, A.: The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London **237**(641), 37 – 72 (1952)
25. Turk, G.: Generating textures on arbitrary surfaces using reaction-diffusion. In: SIGGRAPH ’91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques, pp. 289–298. ACM, New York, NY, USA (1991). DOI http://doi.acm.org/10.1145/122718.122749
26. Weiskopf, D., Ertl, T.: A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In: GI ’04: Proceedings of Graphics Interface 2004, pp. 263–270. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (2004)
27. van Wijk, J.J.: Image based flow visualization. In: SIGGRAPH ’02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 745–754. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/566570.566646
28. van Wijk, J.J.: Image based flow visualization for curved surfaces. In: VIS ’03: Proceedings of the 14th IEEE Visualization 2003 (VIS’03), p. 17. IEEE Computer Society, Washington, DC, USA (2003). DOI http://dx.doi.org/10.1109/VISUAL.2003.1250363
29. Zhang, S., Demiralp, C., Laidlaw, D.H.: Visualizing diffusion tensor mr images using streamtubes and streamsurfaces. IEEE Transactions on Visualization and Computer Graphics **9**(4), 454–462 (2003). DOI http://doi.ieeecomputersociety.org/10.1109/TVCG.2003.1260740
30. Zhang, E., Hays, J., Turk, G.: Interactive Tensor Field Design and Visualization on Surfaces. IEEE Transactions on Visualization and Computer Graphics **13**(1), 94–107 (2007). DOI http://dx.doi.org/10.1109/TVCG.2007.16

31. Zheng, X., Pang, A.: Hyperlic. In: VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03), p. 33. IEEE Computer Society, Washington, DC, USA (2003). DOI http://dx.doi.org/10.1109/VISUAL.2003.1250379