Exploring 2D Tensor Fields Using Stress Nets

Andrew Wilson Rebecca Brannon
Sandia National Laboratories

Abstract

In this article we describe stress nets, a technique for exploring 2D tensor fields. Our method allows a user to examine simultaneously the tensors' eigenvectors (both major and minor) as well as scalar-valued tensor invariants. By avoiding noise-advection techniques, we are able to display both principal directions of the tensor field as well as the derived scalars without cluttering the display. We present a GPU-only implementation of stress nets as well as a hybrid CPU/GPU approach and discuss the relative strengths and weaknesses of each.

Stress nets have been used as part of an investigation into crack propagation. They were used to display the directions of maximum shear in a slab of material under tension as well as the magnitude of the shear forces acting on each point. Our methods allowed users to find new features in the data that were not visible on standard plots of tensor invariants. These features disagree with commonly accepted analytical crack propagation solutions and have sparked renewed investigation. Though developed for a materials mechanics problem, our method applies equally well to any 2D tensor field having unique characteristic directions.

CR Categories: I.3.3. [COMPUTER GRAPHICS]: Picture/Image Generation–Line and curve generation; J.2. [Physical science and engineering]–Engineering

Additional Keywords: tensor field, stress tensor, streamlines, controlled density streamlines, crack propagation

1 Introduction

Tensor visualization is a relatively new area of study compared to vector and scalar field visualization. It is difficult because the meaning of a tensor is strongly problem-dependent. Whereas a vector field usually has an intuitive meaning as a representation either of flow or force, a single tensor can represent such disparate entities as stress forces, strain stretches, the gradient of a velocity field, and diffusion of water within tissue. There are also many aspects of a tensor that can be displayed, including its invariants, eigenvalues, eigenvectors, and even its individual components.

In light of this difficulty, the most successful tensor visualization algorithms so far have been domain-specific. Rather than attempting to display the whole of the tensor data at once, such algorithms extract only the information necessary to the user's area of interest. Examples of such methods include Mohr's circles for stress/strain data [6] and geometric extraction methods for diffusion tensor MRI data [28]. Although the search for a universally useful tensor visualization method continues, we believe that domain-specific methods will always have an advantage in conveying meaning to a user.

Sandia National Laboratories
PO Box 5800, M/S 0822
Albuquerque, NM 87185-0822, USA
{atwilso,rmbrann}@sandia.gov

IEEE Visualization 2005
October 23-28, Minneapolis, MN, USA
0-7803-9462-3/05/$20.00 ©2005 IEEE.

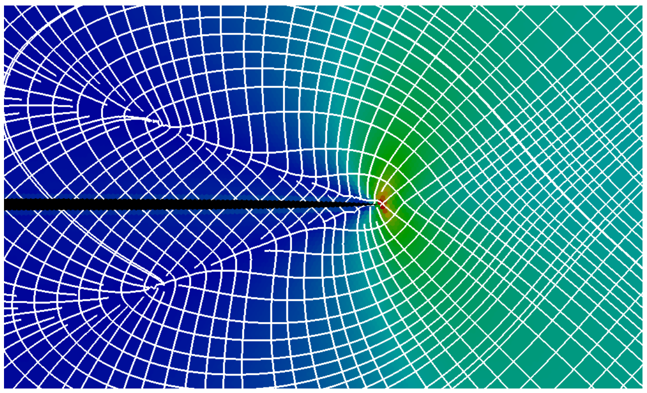


Figure 1: A stress net for one time step in an atomistic simulation of crack propagation. This image shows the directions of maximum shear (the white grid lines) and the magnitude of the deviatoric component of the stress tensor (underlying color).

After completing a simulation of crack propagation at atomic scale, materials scientists within our laboratory sought to compare their results with published asymptotic continuum-mechanics solutions for a nearly equivalent problem. However, the information they needed could not be seen in standard 2D plots of tensor invariants. We were asked to develop a tool with the following capabilities:

1. Display, at all points within the data, the orientation of the maximum shear directions (which are simply 45° rotations of the major and minor eigenvectors)
2. Provide a continuous global view of the data
3. Browse through the data and zoom in for higher-resolution detail
4. Color the data using some scalar invariant derived from the tensors

Contributions: In this paper we describe *stress nets*, a novel visualization algorithm for exploring 2D tensor fields. A stress net can display both major and minor eigenvector fields (or, as needed by our customers, vectors derived from these eigenvectors) as well as some scalar quantity derived from the data. Moreover, our method does not limit the scalar display to hue only, as is commonly the case in noise-based approaches. Although we deal with non-isotropic symmetric tensors in this paper, our method can handle asymmetric tensors as well (with minor revisions). We have incorporated stress nets into an application used by geomechanicists to investigate models of crack propagation. Our application has allowed them to identify discrepancies between the properties of a simulation of crack propagation at atomic scale and the predictions made in the continuum mechanics literature.

The rest of this paper is organized as follows. We survey related work in Section 2. In Section 3, we describe the driving problem in mechanics that led to the development of stress nets. In Section 4 we present two different implementations of stress nets, one using the GPU alone and one using both the CPU and

GPU. In Section 5 we discuss the performance, advantages, and disadvantages of each approach as well as our customers' reaction to the method. In Section 6 we conclude with a brief discussion of possible future work.

2 Previous Work

In this section we survey related work in tensor visualization. Compared with vector visualization, tensor visualization is a relatively unexplored area of study. Moreover, it is difficult to find generally applicable tensor visualization methods because the meaning of a tensor is highly domain-specific. We give examples of both general and domain-dependent visualization techniques.

2.1 General tensor visualization methods

One approach to tensor visualization is to exploit the fact that tensors, like matrices, have eigenvalues and (possibly indeterminate) eigenvectors. In this context, we can treat the eigenvectors as velocity fields and apply generalizations of vector visualization methods. The common hedgehog plot of a vector field can be extended to a field of glyphs. For example, Haber [9] and Kriz et al. [16] construct a field of ellipsoids to represent a field of tensors. The major and minor axes of each ellipsoid are aligned with the major and minor eigenvectors of the tensors and scaled according to the corresponding eigenvalues. While these plots can be useful for local inspection, the problems of clutter, occlusion, and ambiguity in shape make it difficult to observe the behavior of the tensor field over space. De Leeuw and van Wijk [17] use a more complex glyph, the *flow probe,* to illustrate several quantities within a flow field. This conveys more information at the expense of more screen space for each glyph, reducing further the number of glyphs that can be usefully displayed. Hyperstreamlines [7] are an alternative approach that combines aspects of glyphs and flow visualization. Streamlines are traced through the velocity field formed by the major eigenvectors of a set of tensors. Then, an ellipse is swept along each streamline. The minor eigenvectors and eigenvalues of the data are used as the direction and length of the axes of the ellipse. Although this method illustrates changes in the eigenvectors over a path in space, it introduces the common problem of choosing appropriate seed points for the streamlines in order to display the most important features in the data. Moreover, like fields of glyphs, hyperstreamlines encounter problems of clutter and occlusion as more lines are added to the display.

Another approach to conveying the global structure of a tensor field is to decompose it into its topological structure [10,22]. This structure consists of the field's degenerate points, where the tensors have duplicate eigenvalues, and a set of skeleton curves connecting these points. Although this method allows for reconstruction of the tensor field, interpretation can be difficult.

Noise-based vector visualization methods have also been adapted for tensors. HyperLIC [27], a generalization of the line integral convolution algorithm for vector fields [5], integrates a noise field over many small regions of the data. The shape and size of each region is determined by the eigenvectors and eigenvalues of the tensors in that region. The resulting image shows the field of major eigenvectors, smoothed in regions where the tensors are (nearly) isotropic. Hotz et al. [12] take a different approach. They use LIC to generate images for both the major and minor fields of eigenvectors in tensor data, then overlay the two images for display. This is similar in principle to our approach. Their approach to displaying a scalar variable differs from ours in that they use the variable to determine the hue of the LIC image. It can be difficult to separate variability in the luminance of the LIC image from differing values in the scalar being displayed.

2.2 Domain-specific methods

In some cases, the problem domain that gives rise to the tensor data being examined suggests a visualization technique. Exploiting this technique allows us to present an image the users will understand quickly by keying into metaphors they have already learned. For example, Mohr's circles, originally developed around 1900, are commonly taught in undergraduate engineering classes as a way to visualize and interpret stress tensors. The circles provide a visual estimate of the tensor's eigenvalues as well as an overall measure of whether the tensor represents a compressive, tensile, or combined force – all properties that are important in mechanics and materials science. Crossno et al. [6] use Mohr's circles to convey an overview of the forces within a finite-element geomechanical data set. Dickinson [8] addresses stress/strain fields in a more general treatment of interactive methods for scalar, vector, and tensor data. He points out that the orientation components of tensor data are important in their own right and shows an example of a method similar in spirit to stress nets.

Diffusion tensor MRI (DT-MRI) can also produce tensor data with a natural interpretation. The dominant eigenvectors of the diffusion tensors indicate pathways of maximum diffusion, corresponding to structures of interest such as neural fibers within the brain. Methods such as streamtubes [26] and oriented tensor reconstruction [28] work by recognizing and extracting those structures for display. Weinstein and Kindlmann [25] combine a glyph-based approach with direct volume rendering by mapping the anisotropy of diffusion to both hue and the lighting model at each point within the data.

Tchon et al. [21] apply tensor visualization in the context of mesh generation and optimization for finite element simulations. They use the Riemannian metric tensor, a measure of the "best" shape of a mesh element at each point on a surface, and construct two vector fields from the major and minor eigenvectors of the metric tensor. They trace streamlines through these fields to construct a net qualitatively similar to the optimal mesh for a particular metric and data set. Their method differs from ours in that the step size taken at each point during streamline integration is governed by the magnitude of the eigenvalue at that point. This is entirely appropriate for mesh construction and optimization, producing larger elements in regions where the data are smoother, but not as helpful for visual inspection. We achieve similar effects by allowing the user to zoom in and out to view the data and the stress net at different scales.

3 Driving Problem

We developed stress nets in response to a request from a materials scientist studying the propagation of cracks through an elastic-plastic material. The simulation setup for this study is illustrated in Figure 2. We begin with a 2D rectangular slab of some notional material. Tension is applied to the material by pulling vertically on the upper and lower faces A and B. Eventually, the stress near the crack tip exceeds the material's failure threshold. When this happens, a crack at point C will propagate from left to right along the material's center line.

The reason for this study is that there are multiple models in the materials science literature that describe crack propagation. One solution, due to Leighton, Champion and Freund [18], describes cases where the crack is propagating at some non-negligible velocity $\boldsymbol{v}$ (the dynamic case). Another solution, due to Achenbach and Dunayevsky [1], describes the behavior of the material as $\boldsymbol{v}$ vanishes (the quasi-static case). Intuition would suggest that the dynamic solution should converge to the quasi-static one as $\boldsymbol{v}$ approaches zero. This is not the case: the models make qualitatively different predictions about the shear forces within the material. Our customers sought to determine whether

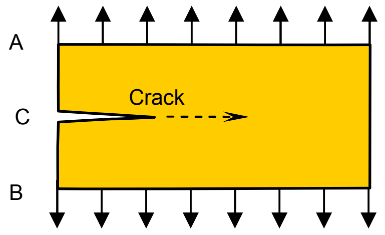


Figure 2: Simulation of crack propagation. A 2D slab of material is stressed by pulling vertically on faces A and B. When the tension exceeds the material's failure threshold, a crack beginning at point C will propagate from left to right.

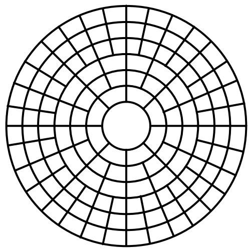


Figure 3: A notional stress net for a circular field. We attempt to keep the size of the cells of the net constant across space to avoid crowding the display.

or not independent atomistic solutions to a nearly equivalent problem could offer insight leading to revisions in the continuum mechanics solutions that might resolve this discrepancy.

3.1 Customer Requirements

Our customers asked for a tool that would allow them to inspect the structure of shear forces within their simulation results. This includes the directions of maximum shear, which are derived from the eigenvectors at each data point, and any one of several scalar quantities derived from the stress tensor. These quantities included the tensor's eigenvalues, its other invariants (such as the trace and the magnitude of the deviatoric part), and its individual components. All of this had to be derived from the raw simulation data, which was supplied as a series of points. Each point represents a single atom and specifies a 2D position in space as well as a symmetric 2D tensor representing the stress forces incident upon that atom. Both the orientations of the shear forces and the underlying scalar field had to be shown simultaneously across the visible extent of the data. The goal of this study was to compare qualitatively the simulation results with the predictions made in [18] and to look especially for abrupt changes of direction in the shear forces near the crack tip. Such changes are not permitted by those predictions. This information does not appear in standard 2D plots of tensor invariants: it is only visible in the orientation portion of the tensors.

3.2 Stress Nets

We chose as a model for our solution a deformed grid whose lines reflect the orientation of the underlying tensor data. Instead of tying the grid's density to the magnitude of the eigenvalues of the tensor field, as is done in [21], we attempt to keep the size of the grid cells roughly constant over the whole of the data. This sort of display is similar in spirit to an electric field diagram including both field lines and equipotential lines, as shown in Figure 3. Moreover, the derived scalar must be clearly visible along with the net itself: the two must not obscure one another. When color is used to display a scalar variable in noise-based methods such as LIC and HyperLIC, the variation in color (due to the scalar variable) and the variation in luminance (due to the noise texture) can be difficult to disambiguate. By creating a grid instead of a space-filling texture, we leave plenty of room free for displaying the scalar variable exclusively. We refer to the orientation component of the display as the *stress net* and the derived scalar variable as the *scalar field*.

We addressed the following issues while developing stress nets:

1. The data are supplied as atoms without extent. How do we construct a space-filling representation that can be used for display or query?
2. How do we construct and render the stress net?
3. How does the system respond to zooming in and out? Should the net be recomputed automatically at each frame?
4. How do we handle situations where the eigenvectors are poorly defined because of repeated eigenvalues?

4 Stress Nets

We have implemented two different versions of stress nets in a tool used in our customers' study. We used VTK [19] for the rendering components of our system and Qt [3] for its user interface. In the rest of this section we describe the two implementations and the design decisions that guide them.

4.1 GPU-only implementation

Our first implementation of stress nets computed both the scalar field and the stress net on programmable graphics hardware. This implementation is separated conceptually into rendering the scalar field and rendering the stress net, although the actual code performs both tasks in a single pass. First we discuss the rendering of the scalar field using a discrete Voronoi diagram of the input data. Next, we show how the stress net is rendered as a texture map on top of the scalar field.

4.1.1 Discrete Voronoi diagram and Scalar Field

We draw the scalar field by building a discrete Voronoi diagram of the input points using the GPU. There are several methods in the literature for accomplishing this. We adopted the method described in [11] for simplicity. At each data point, we draw a cone whose apex points straight up toward the view plane. All cones have the same slope and radius. The radius is chosen empirically so that the only gap visible in the scalar field is in the interior of the crack.

After rendering, we are left with an image where the fragment with the lowest Z value at each pixel is part of the cone belonging to the data point nearest that pixel. The visible portion of each cone is therefore a sampled representation of its data point's Voronoi region. We can use this to render the scalar field simply by using the currently selected scalar to assign a color to each cone.

4.1.2 Drawing the Stress Net

We render the stress net as a texture map on top of the scalar field. Texture coordinates are defined in screen space rather than world space so that the net's resolution will adapt automatically to the visible extent of the data. The user can specify a scaling factor to determine how many times the texture repeats across the screen and thus how fine the net itself is.

The net texture is applied using vertex and fragment shaders. A texture coordinate register is used to pass in the orientation of the stress net for each point in the scalar field. Within each cell, the screen-space texture coordinates are rotated to match this orientation. Since we know that the eigenvectors and hence the shear directions in the original data will always be perpendicular to one another where they exist at all, we can render both grid directions at once. Our particular application contained no

```
StressNetVertexShader( float4 worldPosition,
                       float  netOrientation,
                       float4 netOrigin,
                       float  textureScale,
                       float  windowAspectRatio )
{
   float4 screenPosition = mul(ModelViewProjection,
                               worldPosition);
   float4x4 netRotation =rotateAroundZAxis(netOrientation);

   // Make the grid cells square in screen space
   screenPosition.x *= windowAspectRatio;
   // Normalize after projection
   screenPosition /= screenPosition.w;

   float4 normalizedPosition = screenPosition - netOrigin;
   float4 netTextureCoords = normalizedPosition * netRotation;
   netTextureCoords *= textureScale;

   return normalizedPosition;
}

ApplyStressNetTexture(float4 netTextureCoords,
                      float textureWeight,
                      float4 scalarFieldColor,
                      uniform sampler2D OrientationTexture)
{
   float4 textureColor =
            tex2D(orientationTexture, netTextureCoords);
   return scalarFieldColor + textureWeight * textureColor;
}
```

Figure 4: Pseudocode for vertex and fragment programs that generate the stress net in the GPU implementation.

isotropic regions where the eigenvectors become non-unique because of equal eigenvalues. If isotropic or nearly isotropic states are possible, an appropriate generalization of our method would be to make the opacity of the stress net texture proportional to the magnitude of the stress deviator (hence making the net disappear at isotropic states where the deviator is zero). The choice of the threshold below which the net is completely transparent should be application- and data-dependent in order to show the net only where there is enough anisotropy for the characteristic directions to be meaningful.

Pseudocode for computing and applying the net texture is shown in Figure 4. The texture map used by the fragment shader is shown in Figure 5. In practice, we render both the scalar field and the stress net in a single pass.

4.2 CPU/GPU implementation

Our second implementation focused on the goal of computing a smooth, continuous stress net across the whole of the data at the possible expense of speed and memory. This task requires global knowledge. For the lines of the stress net in one cell to connect to lines in a neighboring cell, the start and end positions of that cell's lines must be known in advance. The same is true for connecting a neighboring cell's net to *its* neighbors, and so on through the entire data set.

The streaming nature of programmable graphics hardware makes it difficult to provide such information. Although an iterative process could be used to adjust the positions of the net incrementally within each cell, this would carry the high cost of copying the frame buffer into texture memory after each step. Rather than incur that overhead, we moved the computation of the stress net back onto the CPU to take advantage of random access to global information. Instead of generating the stress net using texture maps, we treated it as two independent sets of streamlines: one set each for the two sets of directions displayed by the stress net. This is similar in concept to the noise-based approach presented by Hotz et al. [12]

4.2.1 Space-filling representation

As before, we begin with the problem of converting the simulation data into a space-filling representation. This representation will be used as the vector field through which the streamlines composing the stress net will be drawn. Our data

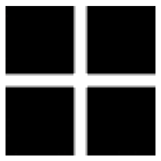


Figure 5: Texture map used for the GPU implementation of the stress net. Black areas of the texture are transparent and allow the underlying scalar field to show through. White areas are opaque.

were provided as a set of points in space, each associated with a symmetric 2x2 tensor. We convert these points into a space-filling representation by inserting them into a data structure that allows nearest-neighbor queries. Once again, we perform no blending of nearby points in order to avoid introducing extra information into the data.

We use a kd-tree for simplicity. There are a plethora of other suitable data structures, including the Voronoi diagram, constrained Delaunay triangulations, a spatial hash table, and even a uniform grid. Our choice of the kd-tree was wholly pragmatic: it is available already within VTK. By selecting this rather than implementing a (potentially) more efficient structure from scratch, we were able to deliver tools to our customers more quickly.

4.2.2 Generating the Stress Net

Once we have a space-filling representation of the data, the construction of the stress net is straightforward. We create two independent sets of streamlines to follow the two different sets of directions in the net. We use Jobard and Lefer's method [13] to help ensure consistent spacing between adjacent streamlines. The user is permitted to control the streamline density by specifying the separation distance as a fraction of screen space. A streamline is terminated if it enters a region where the eigenvectors are ill-defined or if it comes within a threshold distance of another streamline. The streamlines composing the stress net are stored as line segments for later rendering.

Explicit computation and storage of the stress net raises scalability concerns. In order to keep memory requirements low and performance high, we do not recompute the net every time the streamline density or view region changes. Instead, we generate the net only upon request and restrict the region through which it propagates. The streamline region is always centered on the current view and is typically 1.5 times the size of the visible extent of the data. This allows for limited panning and zooming while still covering the visible region with the stress net. Both the extent and the regeneration of the stress net are placed within the user's control.

4.3 Rendering the Combined Display

The scalar field is rendered directly from the input points using the algorithm in Section 4.1.1 in both GPU-only and CPU/GPU implementations. Since the stress net is available as actual geometry (the line segments of the streamlines), we render it on top of the scalar field in a single pass instead of displaying it using textures. Vertex and fragment programs are not necessary in the CPU/GPU implementation.

5 PERFORMANCE AND RESULTS

In this section we discuss the performance, advantages, and drawbacks of the two different implementations of stress nets. All statistics were acquired using one processor of a PC with two 2.6 GHz Pentium IV processors, 4GB of main memory, and an NVIDIA QuadroFX 3000 graphics card with 256MB of memory. The texture map used to display the stress net in the GPU-only implementation took 256KB of this memory.

5.1 GPU-only implementation

Figure 6 shows examples of the output of the GPU-only implementation. The scalar field alone is shown in Figure 6(a). The stress net alone is shown in Fig. 6(b). The two fields combined are shown in Fig. 6(c).

The orientation of the stress net at points across the image can be observed by focusing on the area in question. Since the net is computed in screen space and is recomputed for every frame, zooming into the data permits examination of its orientation at finer scales.

The chief advantages of this implementation are its low overhead and quick startup. Since the stress net is rendered directly from the orientation data within the vertex and fragment programs, there is no need for auxiliary data structures like the kd-tree. Moreover, this implementation will run at the same frame rate regardless of the net's density. This automaticity comes at the expense of slower frame rates, as shown in Table 1.

The major drawbacks of the GPU-only implementation are the rendering artifacts visible in the stress net. We observe two types of artifacts. First, moiré patterns (Figure 11) often become visible when looking at large parts of the data. These occur when the individual cells in the scalar field are much smaller in screen space than the area covered by a single grid cell. In such a situation, the orientation of the net can change drastically between one grid line and the next or even within the area covered by a single grid line. Moreover, the origin about which the grid texture is rotated remains constant across all the data. We made this assumption in an attempt to produce a coherent net in as much of the display as possible. This is a reasonable restriction near the crack tip, which forms the center of rotation for the net in that region. However, it is incorrect in areas such as the wake singularity shown in Figures 1, 7, and 10. We attempted to compute a center of curvature at each data point in order to alleviate this but were unable to achieve useful results due to rapid changes in curvature. In severe cases, such as in Figure 11, the moiré patterns can totally obscure the actual orientations of the stress net.

The moiré artifacts disappear as the user zooms in to view smaller parts of the data. At smaller scales, however, the net appears broken and discontinuous along cell boundaries, as shown in Figure 12. This happens because the vertex program that computes the stress net has no way to ensure the continuity of grid lines from one cell to the next: indeed, the vertex program is not even *aware* of other cells. While it may be possible to introduce such information using recent graphics hardware that permits texture lookups within a vertex shader, we believe that these shortcomings are inherent to the use of a single-pass GPU-only algorithm for computing stress nets.

5.2 CPU/GPU implementation

The hybrid CPU/GPU implementation eliminates nearly all of the artifacts within the stress net as shown in Figures 6 and 7. We observe that the moiré artifacts are no longer present. This is because the lines composing the net are constructed specifically to be continuous and are represented explicitly as geometry instead of existing only as a combination of single-pixel textures. There are still a few discontinuities in the net, produced when grid lines approach too closely to one another and are terminated by the streamline generation algorithm. Also, the spacing between lines (and thus the size of the cells of the net) is not everywhere constant. Nonetheless, the overall display is far more coherent than with the GPU implementation. Rendering speed has also been increased since we no longer recompute the stress net from scratch at every frame.

The greatest advantage of the CPU-based stress net algorithm is that the net's coherence makes it simple to observe the way the stress net changes over a region. We illustrate this in Figure 12. Although both images show the same feature in the data, it is much easier to see in the absence of the discontinuity artifacts present in the GPU-only version.

Table 1: Performance statistics for the GPU and CPU implementations of stress nets on a data set containing 342,000 points and 8 derived quantities. Memory use is measured when the net is visible on the screen with equal density in both implementations. Startup time is the length of time between loading a data set and the appearance of the stress net. The CPU implementation achieves a much higher frame rate because it does not incur the overhead of vertex and fragment programs.

	Memory use	Startup time (sec)	Frame rate (net enabled)	Frame rate (net disabled)
CPU	336M	9	170	270
GPU	252M	1	14.5	14.3

The increased image quality and rendering speed of this approach are balanced by its increased overhead. As seen in Table 1, computing and storing the stress net using the CPU requires more storage space and a substantial startup time. Both of these are due to the kd-tree that we use to look up directions for the stress net during streamline integration. However, there are at least two simple ways to reduce this expense. First, we could substitute some other, more compact spatial data structure for the kd-tree. In the case of the crack-propagation data our customers provided, a geometric hash table or a uniform grid might work well. Second, we could save the kd-tree for a data set to disk after computing it once, then simply reload it during future runs. This would reduce the 11-minute construction time to the few seconds it would take to read the tree from disk. Many different data sets (corresponding to different time steps in the simulation) could be preprocessed in a few hours and then made available for rapid browsing using this approach. It is also possible to optimize the kd-tree implementation itself.

Finally, since we do not recompute the stress net automatically in this implementation, it is possible for the user to outrun it by panning beyond its extent or zooming in or out to the point where its cells are too large or small to be useful. Recomputing the stress net when the user changes either the desired viewing region or the net's density typically takes 5-15 seconds. In practice, we feel that this is not a serious problem. We observe that users tend to spend most of their time examining one small region of the data, then moving rapidly to another area. The brief pause required to recompute the stress net after zooming in or out is small compared to the time spent focusing on an area of interest.

5.2.1 User Reactions

Our customers reacted positively to the results they obtained using our tools. Their first comments did not concern the tool at all, but instead the fact that the structure (now revealed) in their data did not match what the literature predicted. This discrepancy is illustrated in Figures 8-10. This is the goal of any successful visualization tool: to be so transparent that the users see meaning within their data instead of the software used to display it.

After their initial positive response, our customers began to express dissatisfaction with the artifacts in the display. They reported that the moiré patterns and discontinuities in the stress net made it difficult to interpret what was really present in the data. This dissatisfaction led eventually to the hybrid CPU/GPU implementation. The drastic reduction of artifacts in that version enabled our customers to discover a feature in the data (the "wake" of the crack visible in Figures 1, 7, and 10) that had been obscured by artifacts in the GPU-only implementation. They have since asked us to incorporate stress nets into Paraview, an open-source visualization platform used within our laboratory, so that our method can be applied to other data in other domains.

6 Conclusions and Future Work

We have presented stress nets, a novel visualization algorithm for fields of 2D tensors, and discussed the advantages and disadvantages of two different methods for generating and rendering the net. Our first implementation, using only the GPU, automatically recomputed the net to fit the visible extent of the data exactly. However, the streaming nature of programmable graphics hardware resulted in artifacts that ultimately proved unacceptable to our customers. A second implementation using the GPU for the scalar field and the CPU for the stress net yielded faster rendering and much higher image quality at the cost of increased preprocessing time and memory overhead.

We incorporated stress nets into a tool used to explore the results of a simulation of crack propagation. This tool allowed our customers to identify discrepancies between their simulation results and the predictions made by the prevailing model in the literature. Future investigation of these discrepancies may lead to unifying revisions of conflicting continuum solutions and/or changes in the theory by which interatomic forces are converted to stress tensors in atomistic simulations. We consider the use of our tools to advance a completely separate area of science to be a validation of the utility of our approach.

Stress nets are applicable to a broader range of tensor visualization problems, including nearly any problem that involves a symmetric tensor field or, with some revision, even non-symmetric tensor fields as well. Within the geomechanics arena, drilling paths for oil wells must closely follow the eigenvectors associated with maximum stress to ensure well-bore viability. Stress nets can provide a simple and intuitive view of these preferred paths. In biomechanics, we could apply stress nets to visualize the stresses and strains operating within the heart, and "diffusion nets" can be generated in perfect analogy with stress nets. As described here, our method applies to *any* 2D symmetric tensor field.

Stress nets may be extended to support 2D non-symmetric tensor fields whose eigenvectors are not orthogonal. The key is to compute a set of lines for the major and minor fields of eigenvectors separately, then overlay them on one another for display. In the GPU implementation, the net texture would change to a line segment instead of a cross. Two separate texture lookups would be required: one for each field of eigenvectors. The fragment program would combine both textures with the scalar field just as it currently does the single net texture. The CPU/GPU implementation needs no modification to its algorithm, as its two sets of streamlines are already computed separately using the major and minor eigenvectors.

These examples raise the question of how best to extend our method into three dimensions. The simplest approach may be to draw the net on a cutting plane or some other surface. A stress net itself can be computed in three (or more) dimensions with little change to the algorithm, but visualizing the results presents a challenge. We note that the problem of robust visualization methods for 3D vector flow fields is still an active area of research. We could also compute stress nets at several different resolutions simultaneously using a method similar to the one presented by Jobard and Lefer [14] to reduce the number of times the net must be recomputed during viewing.

Finally, our method does not preclude the simultaneous use of other tensor visualization methods. A stress net could be overlaid on a topological decomposition of a tensor field, such as the methods described by Hesselink et al., or used in concert with an alternate display method such as Mohr diagrams. We feel that multiple linked views of a single data set are a promising area for future research.

The authors are grateful to Jon Zimmerman of Sandia National Laboratories for providing the atomistic simulation data and to the anonymous reviewers for their helpful comments. The Department of Energy (DOE) Mathematics, Information, and Computer Science Office funded the visualization portion of this work. This work was performed at Sandia National Laboratories, a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

References

[1] J.D. Achenbach and V. Dunayevsky, "Crack-Tip Plasticity for Rapid Crack Propagation", in Advances in Fracture Research (Fracture 81) 5, ICF5, pp. 2205-2213, 1981.

[2] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. "Anisotropic Polygonal Remeshing". in ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003), vol. 22, ACM Press, San Diego, California, pp. 485-493, July 2003.

[3] Jasmine Blanchett and Mark Summerfield, "C++ GUI Programming with Qt 3", Prentice Hall, January 2004.

[4] Ed Boring and Alex Pang. "Interactive deformations from tensor fields." in Proceedings of IEEE Visualization 1998, IEEE Computer Press, Research Triangle Park, North Carolina, pp. 297-304, October 1998.

[5] Brian Cabral and Leith Leedom. "Imaging vector fields using line integral convolution", in Computer Graphics (SIGGRAPH 1993 Proceedings), vol. 27, ACM Press, pp. 263-272, August 1993.

[6] Patricia Crossno, David H. Rogers, Rebecca M. Brannon, David Coblentz, and Joanne T. Fredrich. "Visualizing of Geologic Stress Perturbations using Mohr Diagrams". To appear in IEEE Transactions on Visualization and Computer Graphics.

[7] Thierry Delmarcelle and Lambertus Hesselink, "Visualizing second-order tensor fields with hyperstreamlines", IEEE Computer Graphics and Applications, vol. 13, IEEE Computer Press, pp. 25-33, July 1993.

[8] Robert R. Dickinson, "A unified approach to the design of visualization software for the analysis of field problems", in Three-Dimensional Visualization and Display Technologies, SPIE Proceedings, vol. 1083, pp. 173-180, January 1989.

[9] Robert B. Haber, "Visualization techniques for engineering mechanics," in Computing Systems in Engineering, 1:37-50, 1990.

[10] Lambertus Hesselink, Yuval Levy, and Yingmei Lavin, "The topology of symmetric, second-order 3D tensor fields", in IEEE Transactions on Visualization and Computer Graphics, 3(1):1-11, January/March 1997.

[11] Kenneth Hoff III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha, "Fast Computation of Generalized Voronoi Diagrams using Graphics Hardware", in Computer Graphics (SIGGRAPH '99 Conference Proceedings), ACM Press, Los Angeles, California, pp. 277-286, August 1999.

[12] Ingrid Hotz, Louis Feng, Hans Hagen, Bernd Hamann, Kenneth Joy, and Boris Jeremic. "Physically Based Methods for Tensor Visualization", in Proceedings of IEEE Visualization 2004, IEEE Computer Press, Austin, Texas, pp. 123-130, October 2004.

[13] Bruno Jobard and Wilfrid Lefer, "Creating evenly-spaced streamlines of arbitrary density", in 8th Eurographics Workshop on Visualization in Scientific Computing, Boulogne-sur-Mer, France, pp. 45-55, April 1997.

[14] Bruno Jobard and Wilfrid Lefer, "Multiresolution flow visualization", in Proceedings of the 9thth International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG '01), Plzen, Czech Republic, February 2001.

[15] Gordon Kindlmann, "Superquadric Tensor Glyphs". in O. Deussen, C. Hansen, D. A. Keim, and D. Saupe, editors, Joint Eurographics -

IEEE TCVG Symposium on Visualization 2004, Konstanz, Germany, 2004.

[16] Ron D. Kriz, Edward H. Glaessgen, and J. D. MacRae, "Visualization blackboard: Visualizing gradients in composite design and fabrication", in IEEE Computer Graphics & Applications, 15(6):10-13, IEEE Computer Press, November 1995.

[17] Willem C. de Leeuw and Jarke J. van Wijk, "A probe for local flow field visualization". in Proceedings of the 4th conference on Visualization '93, IEEE Computer Press, pp. 39-45, 1993.

[18] J.T. Leighton, C.R. Champion, and L. B. Freund, "Asymptotic analysis of steady dynamic crack growth in an elastic/plastic material", in J. Mech. Phys. Solids 35 (1987), 541.

[19] Will Schroeder, Ken Martin and Bill Lorensen, "The Visualization Toolkit", Pearson Education, Inc., 2002.

[20] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis, "OpenGL Programming Guide, Fourth Edition", Addison-Wesley, Boston, Massachusetts, 2004.

[21] Ko-Foa Tchon, Julien Dompierre, Marie-Gabrielle Vallet, and Ricardo Camarero, "Visualizing Mesh Adaptation Metric Tensors", in Proceedings of the 13th international Meshing Round Table, Sandia National Laboratories, Williamsburg, Virginia, pp. 353-363, 2003.

[22] X. Tricoche, G. Scheuermann, and Hans Hagen, "Tensor topology tracking: A visualization method for time-dependent 2D symmetric tensor fields", in A. Chalmers and T.-M. Rhyne, editors, EG 2001 Proceedings, volume 29)3) of Computer Graphics Forum, Blackwell Publishing, pp. 461-470, 2001.

[23] Greg Turk and David Banks, "Image-guided streamline placement", in Proceedings of SIGGRAPH 96, ACM Press, New Orleans, Louisiana, pp. 453-460, August 1996.

[24] Vivek Verma, David Kao, and Alex Pang, "A flow-guided streamline seeding strategy", in Proceedings of IEEE Visualization 2000, IEEE Computer Press, Salt Lake City, Utah, pp. 163-170, 2000.

[25] David M. Weinstein, Gordon L. Kindlmann, and Eric C. Lundberg. "Tensorlines: Advection-diffusion based propagation through diffusion tensor fields", in Proc. of IEEE Visualization '99, IEEE Computer Press, San Francisco, California, pp. 249-253, October 1999.

[26] Song Zhang, Charles T. Curry, Daniel S. Morris, and David H. Laidlaw. "Streamtubes and streamsurfaces for visualizing diffusion tensor MRI volume images", in IEEE Transactions on Visualization and Computer Graphics, vol. 9, n.4, pp. 454-462, October 2003.

[27] Xiaoqiang Zheng and Alex Pang, "HyperLIC", in Proceedings of IEEE Visualization '03, IEEE Computer Press, Seattle, WA, pp. 249-256, October 2003.

[28] Leonid Zhukov and Alan Barr. Oriented Tensor Reconstruction: Tracing Neural Pathways from Diffusion Tensor MRI. in Proceedings of IEEE Visualization 2002, IEEE Computer Press, Boston, Massachusetts, pp. 387-394, October 2002.

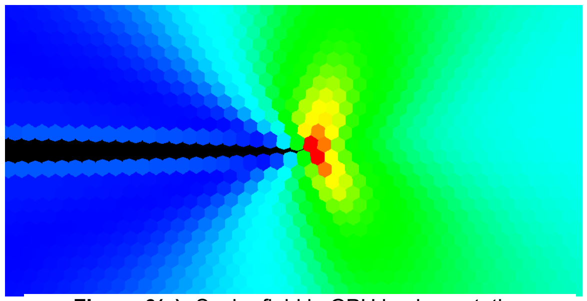


Figure 6(a): Scalar field in GPU implementation

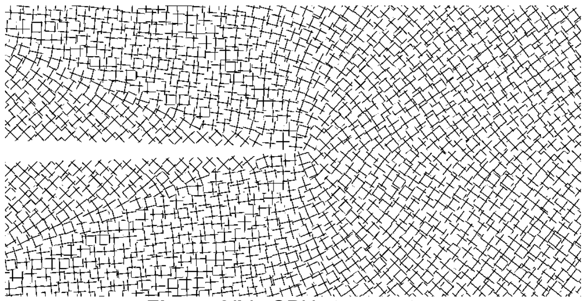


Figure 6(b): GPU stress net

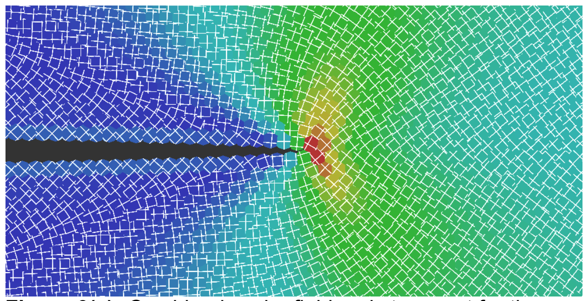


Figure 6(c): Combined scalar field and stress net for the GPU-only implementation. Although it is possible to discern the orientation of the stress field at any point, the rendering artifacts make it difficult to observe its behavior over a wide region.

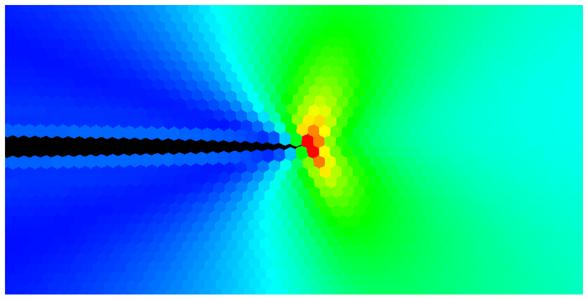


Figure 7(a): Scalar field in CPU/GPU implementation

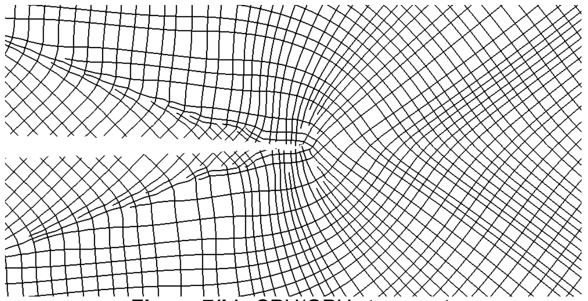


Figure 7(b): CPU/GPU stress net

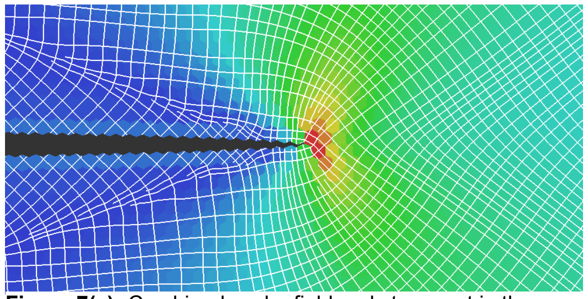


Figure 7(c): Combined scalar field and stress net in the CPU/GPU implementation. Computation of the stress net itself is moved back onto the CPU. The rendering artifacts have disappeared, leaving the features of the data much clearer at the cost of some additional preprocessing.

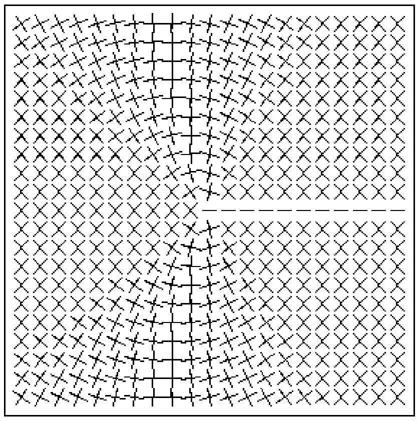


Figure 8: This style of orientation plot was in use by our customers prior to our development of stress nets. Each glyph corresponds to a single data point. The crack tip is in the center of the image. The discontinuous glyphs make it difficult to discern the structure of the changes in the tensor field.

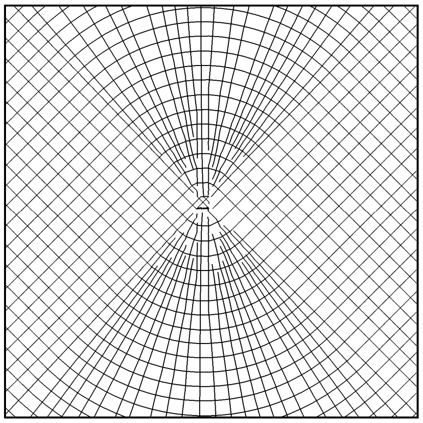


Figure 9: Stress net created from the tensor field shown in Figure 8. This is the asymptotic, analytical solution in the literature. Our customers expected to see this structure.

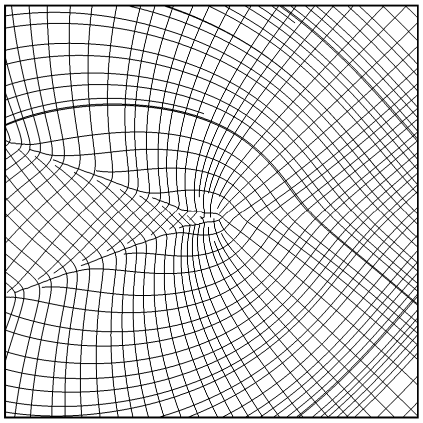
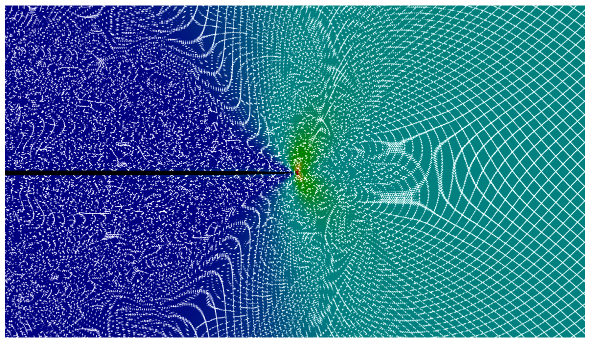
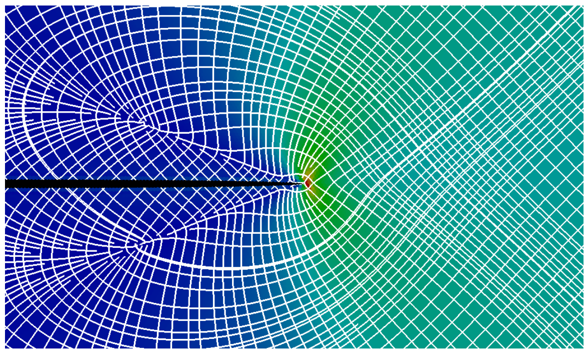


Figure 10: Stress net created from atomistic simulation data. It differs substantially from the predicted structures. The angles around the crack tip (again, in the center of the image) differ from predictions. The structures in the crack's wake, spreading diagonally up and down to the left of the crack, are not present at all in the analytical solution.

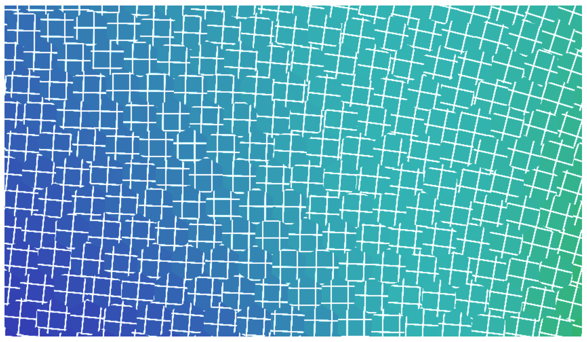


GPU implementation

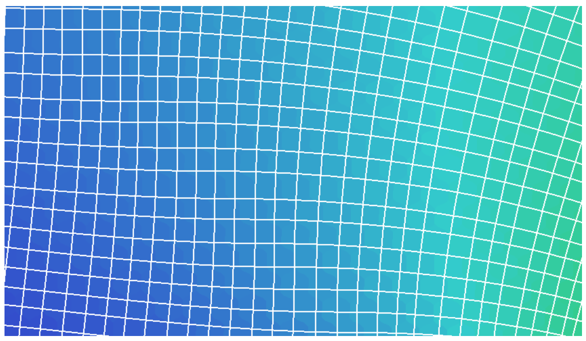


CPU/GPU implementation

Figure 11: Moiré artifacts appear in the GPU implementation when the cells of the stress net are large compared to the individual data cells. This is an extreme example. These artifacts are not present in the CPU/GPU implementation.



GPU implementation



CPU/GPU implementation

Figure 12: When the user zooms into the data, the moiré patterns in the GPU implementation disappear but the stress net appears to break. This happens because the GPU has no way to ensure continuity of the net from cell to cell. The CPU implementation remains continuous at all scales.