Volume Graphics (2006)
T. Möller, R. Machiraju, T. Ertl, M. Chen (Editors)

Interactive 3D seismic fault detection on the Graphics Hardware

Won-Ki Jeong[1†], Ross Whitaker[1], Mark Dobin[2]

[1]School of Computing, University of Utah
[2]ExxonMobil Upstream Research Company

Abstract

This paper presents a 3D, volumetric, seismic fault detection system that relies on a novel set of nonlinear filters combined with a GPU (Graphics Processing Unit) implementation, which makes interactive nonlinear, volumetric processing feasible. The method uses a 3D structure tensor to robustly measure seismic orientations. These tensors guide an anisotropic diffusion, which reduces noise in the data while enhancing the fault discontinuity and coherency along seismic strata. A fault-likelihood volume is computed using a directional variance measure, and 3D fault voxels are then extracted through a non-maximal-suppression method. We also show how the proposed algorithms are efficiently implemented with a GPU programming model, and compare this to a CPU implementation to show the benefits of using the GPU for this computationally demanding problem.

Categories and Subject Descriptors (according to ACM CCS): I.4.0 [Image Processing and Computer Vision]: General; I.3.8 [Computer Graphics]: Applications

1. Introduction

The analysis of seismic data is important for understanding the earth's subsurface and is an important aspect of oil and gas exploration. Because of the growing volume and resolution of seismic data, image processing is becoming an component of this process. Various image processing techniques have been developed over several decades, but most techniques are only semiautomated, because seismic features are so subtle that they require a manual interpretation process by professionals for accurate analysis. In addition, 3D seismic datasets are also very large, and thus interactive processing is typically limited to very simple analysis or to slice-by-slice (2D) processing.

Recently, GPUs have gained popularity in the computer graphics community not only for traditional graphics applications but also for general-purpose computations [OLG*05]. Due to their parallel architecture and extremely wide memory bandwidth, state-of-the-art GPUs have begun to outperform CPUs in raw computational power, allowing researchers to utilize GPUs for computationally demanding problems in several application domains. Despite these advances applications of GPU processing to 3D image processing and geoscience have not yet been fully investigated. The motivation of this work comes from the observation that the time-consuming seismic data analysis can benefit from the modern inexpensive but powerful GPUs, allowing users the freedom of tuning parameters at interactive rates. The main contributions of the paper include efficient GPU implementations of existing algorithms, (i.e. structure tensor analysis, nonlinear PDEs for denoising, and hysteresis thresholding), a novel 3D directional anisotropic diffusion algorithm based on the orientation of the seismic strata, and an application of these techniques and technologies to seismic data interpretation. This paper demonstrates the benefits of using GPUs and user interaction for computationally demanding image processing applications.

2. Previous work

The related work falls into two categories: seismic image analysis and GPU-based processing. Traditionally seismic analysis has been done by manual interpretation of processed 2D slices. With the advent of the rapid increasing computational power, direct processing of 3D seismic vol-

† wkjeong@cs.utah.edu

ume is becoming more practical. The coherence cube, proposed by Bahorich et al [BF95], is an early and important contribution to the detection of faults and other geological features in 3D seismic volumes. In [BF95], coherency is measured by the geometric mean of maximum time-lagged cross-correlation along x and y directions. Because voxel intensities indicate sharp contrasts across fault surfaces, those regions become distinct in the coherence cube. Marfurt et al. [MKFB98] propose a robust coherence estimation algorithm based on multiple traces with locally adapted similarity (or *semblance*) measure. Another variant of coherence cube, based on eigenanalysis of covariance matrix, is proposed by Gersztenkorn et al. [GM99]. A practical survey of several variants of the coherence cube algorithm can be found in [Cho02]. Cohen et al. [CC02] propose a more efficient discontinuity measure computation method using a normalized trace of a small correlation matrix. Lu et al. [LLZ*05] employ higher-order-statistics and supertrace technique for more accurate coherence estimation.

Some oriented filtering methods have been successfully employed to enhance a fault structure [Wei99, BvVV99, FH03] where actual fault surfaces are not required. Three-dimensional filtering methods shown above have given more insights to interpreters, but the methods are usually time-consuming and require significant manual interpretation. Interpretation of the coherence or discontinuity volume is typically a painstaking process. Hence, several semiautomatic fault detection methods have been developed. An early work by Randen et al. [RPSS99] propose a fault response as a projection of the gradient vector onto the local orientation plane, which is measured using a least-square axis fitting method [BGW91]. Bakker introduce a structure tensor for robust orientation computation [Bak02]. Oriented Kuwahara-type fault enhancing anisotropic filtering [BvVV99] has also been applied to enhance fault discontinuity. Eigenanalysis of a structure tensor gives the fault measure of confidence, and fault surfaces are created using a non maximal suppression method. This strategy produces robust results but anisotropic filtering and eigenanalysis require heavy computation. Gibson et al. [GST03] propose a structure tensor approach, similar to [Bak02], but the semblance value is estimated using a user-defined oriented window. A major difference from [Bak02] is that the method creates 3D geometry instead of binary voxels. Pedersen et al. [PRSS02] borrow an idea from a behavior of a group of social insects to enhance fault responces. A recent work by Jacquemin et al. [JM05] uses a Hough transform, one of the traditional feature detection algorithms, to automatically extract 3D fault surfaces.

Because of its highly parallelized architecture and wide memory bandwidth, the GPU has become a promising technology for solving computationally demanding numerical problems. Rumpf et al. [RS01, SR01] first describe the application of graphics hardware for solving partial differential equation (PDE) based computer vision problems. Lefohn et al. [LKHW03] propose an implementation of virtual memory management on the GPU to solve PDEs using the level set method. Goodnight et al. [GWL*03] describe a 2D multigrid solver on the GPU, and several researchers [BFGS03, KW03] describe GPU solvers for sparse linear systems. Several researchers describe particle simulation engines on the GPU [KKW05, KKKW05, KSW04]. Accelerating traditional image processing techniques using the graphics hardware are given in [HE99, VKG03]. An extensive survey of the general purpose computation on the GPU can be found in [OLG*05]. In contrast to its huge popularity in the computer graphics field, very few efforts have been made in the geophysics/geoscience field using the GPU. There is evidence that industrial concerns have recently introduced GPU-based computation for visualization and oil reservoir simulation, but to our knowledge, these results have not been published as research literature.

This paper presents a new method for detecting faults in 3D seismic data. The approach, unlike many others, is fully three-dimensional, including the processing, feature detection, and visualization. To offer the computational challenge of this fully 3D approach, we propose a 3D implementation, which allows processing of moderate sized volumes at interactive rates. The GPU implementation requires fast evaluations of 3D partial differential equations, voxel-wise eigen analysis, and interactive rendering.

3. Overview of the fault detection system

The goal of this work is to extract faults from the raw input seismic volume (3D image) in a semi-automatic way (Fig 1 shows a 2D slice of such data). Because faults occur where two crustal blocks slip or move against each other, we can detect these faults by finding for discontinuities along each layer of the seismic strata, as shown in Fig 1c.

(a) Input image (b) Faults (c) Schematic example

Figure 1: *Example of fault detection from seismic data.*

The proposed fault detection method, based on strategies described in the literature from the previous section, consists of four steps as follows:

`Step 1` Compute local orientation of strata using structure tensor analysis (Section 4).

`Step 2` Apply directional anisotropic diffusion to remove noise while retaining and enhancing faults (Section 5).

Step 3 Compute a fault-likelihood volume, a scalar volume that represents how likely each point belongs to a fault, using a directional variance measure (Section 6).

Step 4 Extract one-voxel thick fault surfaces using a non-maximal-suppression method. This includes hysteresis thresholding to keep only strong features or features connected to features (Section 6).

The proposed method, as with many other approaches to this problem, includes a number of free parameters, that must be tuned throughout the processing pipeline. Each step must be repeated until proper parameter values are found. The time and contrast parameters for the diffusion in Step 2 must be properly chosen to optimize detection. The window size of variance computation in Step 3 affects the sensitivity of the fault detection. High and low thresholds for hysteresis thresholding in Step 4 should be chosen properly to get less noisy and detailed fault surfaces. Tuning such parameters can be a time-consuming process, but the proposed GPU-based fault detection system enables this process in an interactive manner. In the following sections, each step will be explained in detail.

4. Structure tensor for seismic image analysis

The so-called *seismic horizon* is rarely horizontal, which means that a plane tangent to a seismic strata is usually generally *not* parallel to *x*-*y* (ground) plane. Tracking seismic horizon orientation is important because the filtering and fault detection should be done using local neighborhoods. To measure the orientation of seismic strata we use a structure tensor, a robust measure of local orientation of flow-like structures that has been actively studied and used for various applications in image processing and computer vision [Wei99, Bak02, FH03].

A structure tensor J, which represents a local orientation in an n-dimensional space (for seismic $n = 3$), is defined by a tensor product of a n-dimensional vector x as follows.

$$J = x \otimes x \tag{1}$$

where $x \otimes x = xx^T$. The local orientation is generally defined using a gradient vector. The tensor formulation allows the averaging of orientations in a way that is insensitive to sign (the equivalence of vectors pointing in opposite directions). Because the structure tensor represents an *orientation* rather than a *direction*, blurring tensors (e.g. using Gaussian convolution) provides a more robust orientation field, because it allows gradient vectors in opposite directions to support rather than counteract one another. Thus, the analysis of the structure tensor proceeds by blurring the tensor field and extracting the nonzero eigenvalues of the tensor. We define a structure tensor J of the gradient of an image I as follows [Wei99]:

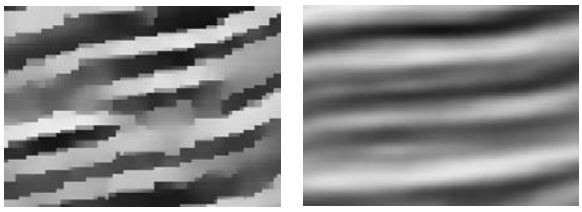
$$J_\rho(\nabla I_\sigma) = K_\rho \star (\nabla I_\sigma \otimes \nabla I_\sigma) \tag{2}$$

where $K_\rho(x) = \frac{1}{\sqrt{2\pi}\rho} e^{-\frac{x^2}{\rho^2}}$, $I_\sigma = K_\sigma \star I$, and $\star$ is a convolution operator.

The eigenanalysis of the matrix J_ρ provides information about the aggregate orientation and local coherence of gradient information. Eigenvectors of the structure tensor matrix form an orthogonal local coordinate system. Corresponding eigenvalues represent the strength of the direction along the eigenvector because Eq 1 is not normalized. For 3D input, the structure tensor J is a 3×3 positive semi-definite matrix, whose eigenvalues and eigenvectors can be found quickly using an analytic method [HBPA01]. For 3D seismic data, the dominant eigenvector represents the gradient direction, orthogonal to the seismic horizon, while the other two eigenvectors form a plane orthogonal to the gradient which is parallel to the strata in the data. Because a fault is a discontinuous region on a seismic horizon, we can assume that the second largest eigenvector, which is the direction of the largest change in the seismic horizon plane, points across the fault plane.

5. Directional anisotropic diffusion

In the last two decades a great deal of image processing has focused on nonlinear processes that reduce noise while preserving features. A significant body of work has addressed the generalization of the approach of Perona and Malik [PM90], who propose a nonlinear PDE which entails a variable conductance diffusion, for edge-preserving denoising. With some modifications the method can be applied to seismic data. The modification stems from the observation that seismic data includes wavelike patterns across (perpendicular to) the strata that result from the oscillating sound waves interacting with the transitions in material (e.g. rock) densities. A direct application of Perona and Malik diffusion results in undesirable artifacts in the form of blocky boundaries between horizons (Fig 2 a).



(a) Perona et al. (b) Coherence enhancing

Figure 2: *Comparison of Perona et al.'s and coherence enhancing diffusion applied to a seismic image*

Weickert [Wei99] has proposed an anisotropic diffusion process for images, called *coherence enhancing diffusion*, which directs the diffusion only along the directions defined by the structure tensor (Fig 2 b). However, coherence enhancing diffusion does not preserve edge-based features. To

remove noise while enhancing both fault discontinuity and coherence along horizons, we modify the the coherence enhancing flow, using an edge-based term as follows:

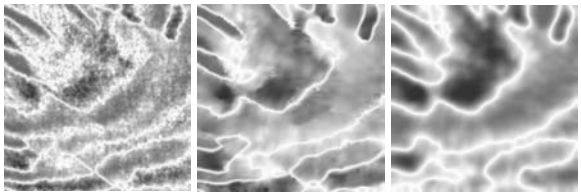
$$\frac{\partial I}{\partial t} = \nabla \cdot (D \nabla I) \tag{3}$$

where

$$D = \begin{bmatrix} \vdots & \vdots & \vdots \\ v_1 & v_2 & v_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} \cdots & v_1 & \cdots \\ \cdots & v_2 & \cdots \\ \cdots & v_3 & \cdots \end{bmatrix}$$

, and $\lambda_1 \approx 0$, $\lambda_i = e^{-\frac{(v_i \cdot \nabla I)^2}{k^2}}$ for i=2,3, and v_i is i-th eigenvector of the structure tensor matrix J.

The main idea behind Eq 3 is to take the benefits from both [PM90] and [Wei99] by applying Perona and Malik diffusion along seismic horizon to enhance coherency along the strata while detecting discontinuity across faults on the horizon plane. A similar idea has been shown in [FH03], based on the scalar continuity factor defined by the difference of traces of the structure tensor matrix before and after averaging using a Gaussian. However, the proposed method uses a new definition of the diffusion tensor D so that the anisotropic diffusion is aligned to the local seismic orientation. The method also incorporates the contrast parameter k, which allowing users to freely adjust the sensitivity of detecting faults. The method of [FH03] does not allow for contrast and is, in our experience, more sensitive to noise. Figure 3 shows results of the proposed method with various contrast parameters.



(a) Input image (b) k=6 (c) k=100

Figure 3: *Directional anisotropic diffusion with varying k.*

6. Fault extraction

Fault detection in 3D seismic data is similar to edge detection in conventional image processing. However, a simple axis-aligned gradient-based edge detector will pick not only faults but also layers across seismic strata. In addition, seismic datasets are usually noisy, and thus the unprocessed gradient is especially prone to false positives. Hence, we need to use a fault detector that is more robust to noise and adjustable to local orientation. We have tested several edge detectors and have concluded that directional variance is the best overall choice. The variance-base fault detector is based on the observation that each seismic horizon consists of points having similar intensity values unless faults are presented. We define *fault-likelihood* of a point x as an average variance of the voxel intensities in a user-defined window centered at x. If we compute fault-likelihood on every voxel of the volume, we get a *fault-likelihood volume*.

Once the local orientation is computed on each point, we can create a fault-likelihood volume through two steps. In the first step, the directional variance V, the variance of points on the seismic horizon plane, is computed on every voxel p. The horizon plane is defined by the second and third eigenvectors of the structure tensor at p. The number of points for computing the variance, the window size, is defined by the user. In the second step, the fault-likelihood F, defined as the average of the directional variances along the positive and negative dominant eigenvector direction, is computed on every voxel p. The following definition uses $2n+1$ samples to compute the fault-likelihood value for each point.

$$\begin{aligned} V(p) &= Var(plane_p) \\ F(p) &= \frac{1}{2n+1}\left(\sum_{i=-n}^{n} V(p+iv_1)\right) \end{aligned}$$

where $plane_p$ is the horizon plane centered at p.

Once we have a fault-likelihood volume, we can extract fault surfaces by applying a non-maximal-suppression (NMS) and a hysteresis thresholding, in a way similar to that used in the Canny edge detector [Can86]. For edge detection in images, the strategy is to find a local maximum of the gradient magnitude the image in the direction of the gradient vector. A fault-likelihood volume serves as a gradient image in our algorithm, and therefore applying NMS gives the voxels located on the fault surfaces. For faults the orientation of the second eigenvector give the direction across the fault. Applying Gaussian smoothing on the fault-likelihood volume before applying a NMS helps to suppress small local maxima.

A single global threshold is harder to use on the dataset having many false positives. Therefore, we propose a hysteresis thresholding, which is a two level thresholding, to capture feature details while filter out false positives. The user provides upper and lower thresholds. If a voxel intensity is greater than the upper threshold, we keep it as a feature. If the intensity is smaller than the lower threshold, then we discard it. If the intensity is between upper and lower thresholds, then keep it as a feature only if the voxel is topologically connected to any voxel having the intensity greater than the upper threshold. Hysteresis thresholding greatly improves the fault detection results. The implementation details are given in section 7.2.

7. GPU implementation

This section explains how the algorithms given above can be efficiently implemented on the GPU. More information on GPU programming can be found in [HLB*05].

We use a set of 2D textures to represent a 3D volume on the GPU because rendering to a 3D texture is not yet supported by many recent graphics cards. Hence, a single render pass on a volume is a set of render passes, one pass per slice. We use 32 bit floating-point textures as intermediate buffers to prevent precision errors when solving the nonlinear PDE system for the diffusion. Computation on the GPU is mapped to a rendering process defined as binding the source textures and then rendering a screen-size quad onto a target texture using vertex/fragment programs. The GL_EXT_framebuffer_object extension enables us to render directly to textures without reading back to/from CPU memory.

7.1. Directional anisotropic diffusion

Multiple render targets are used to store three eigenvectors in separate textures for future use. Eigenvalues are computed by finding solutions of a cubic equation, and corresponding eigenvectors are computed using the analytic method given in [HBPA01]. The following code is the GPU implementation of the eigenanalysis of a 3x3 tensor matrix using Cg language. Six elements (tensor is symmetric) of the tensor matrix are read from Tex0 and Tex1, and three computed eigenvectors are written to col0, col1, and col2.

```
void eigenanalysis(out float4 col0 : COLOR0,
                   out float4 col1 : COLOR1,
                   out float4 col2 : COLOR2,
                   float2 uv0 : TEXCOORD0,
                   uniform samplerRECT Tex0 : TEXUNIT0,
                   uniform samplerRECT Tex1 : TEXUNIT1)
{
   float3 eval, evec[3], J[2];

   // Reading 6 Structure Tensor Elements
   J[0] = texRECT(Tex0, uv0).xyz;
   J[1] = texRECT(Tex1, uv0).xyz;

   // Compute Eigenvalues
   float b = -(J[0].x + J[1].x + J[1].z);

   float c = J[0].x * J[1].x + J[1].x * J[1].z +
             J[0].x * J[1].z - J[0].y * J[0].y -
             J[1].y * J[1].y - J[0].z * J[0].z;

   float d = J[0].y * J[0].y * J[1].z +
             J[1].y * J[1].y * J[0].x +
             J[0].z * J[0].z * J[1].x -
             J[0].x * J[1].x * J[1].z -
             J[0].y * J[1].y * J[0].z -
             J[0].z * J[0].y * J[1].y;

   float f = (3.0*c - b*b)/3.0;
   float g = (2.0*b*b*b - 9.0*b*c + 27.0*d)/27.0;
   float h = (g*g)/4.0 + (f*f*f)/27.0;

   if(f == 0 && g == 0 && h == 0) {
     eval.x = eval.y = eval.z = -pow(d,1.0/3.0);
   }
   else {
     float i = sqrt((g*g)/4.0 - h);
     float j = pow(i, 1.0/3.0);
     float k = acos(-(g/(2.0*i)));
     float m = cos(k/3.0);
     float n = sqrt(3.0)*sin(k/3.0);
     float p = -(b/3.0);
     eval.x = 2.0*j*cos(k/3.0) - (b/3.0);
     eval.y = -j*(m-n)+p; // swapped
     eval.z = -j*(m+n)+p;
   }

   // Compute Eigenvectors
   float3 A, B, C, D, E, F;

   A = J[0].xxx - eval;
   B = J[1].xxx - eval;
   C = J[1].zzz - eval;
   D = J[0].yyy*J[1].yyy - B*J[0].zzz;
   E = J[0].zzz*J[1].yyy - C*J[0].yyy;
   F = J[0].zzz*J[0].yyy - A*J[1].yyy;

   evec[0] = D*E;
   evec[1] = E*F;
   evec[2] = D*F;

   col0.xyz = normalize(float3(evec[0].x,evec[1].x,evec[2].x));
   col1.xyz = normalize(float3(evec[0].y,evec[1].y,evec[2].y));
   col2.xyz = normalize(float3(evec[0].z,evec[1].z,evec[2].z));
}
```

We employ the explicit Euler integration scheme to solve Eq 3, which implements a variable conductance diffusion along the second and third eigenvector directions respectively. We use trilinear interpolation, defined as $I_{nb(i)}$ below, to compute directional derivatives. The update scheme is defined as follows:

$$I_{new} = I_{old} + dt\Big(\sum_{i=1,..,4} (I_{nb(i)} - I_{old}) e^{-\frac{(I_{nb(i)} - I_{old})^2}{k^2}}\Big) \qquad (4)$$

where $I_{nb(i)}$ is a trilinear interpolation from the current point along the direction v_2, $-v_2$, v_3, and $-v_3$ for $i = 1,...,4$ respectively.

For each render pass, the input to the shader is the textures storing the eigenvectors and the current image, and the output is a new image updated by Eq 4. We use two volumes for ping-pong rendering, which means that one volume is the source and the other is the target to avoid reading from and writing to the same texture. We repeat this rendering until the user-defined time, or the number of iterations, is reached. The time step dt is set to 0.25 in our experiments for the stable computation.

7.2. Fault detection and thresholding

A fault-likelihood volume is created through two render passes, one is computing directional variances and the other is averaging variances to get fault-likelihood values. We use $(2n+1)^2$ sample points for computing the directional variance, where n is given by the user. To compute the fault-likelihood volume, we sample $2n+1$ directional variances along the gradient direction and average them. Usually n is not zero, meaning that distant neighborhoods are needed to compute directional variances and fault-likelihood values. Hence, we use a 16-bit floating (half float) 3D texture to utilize a hardware trilinear interpolation.

For hysteresis thresholding, we cannot use recursive functions as is typical in a conventional (CPU) implementation. Instead the GPU implementation uses an iterative method by checking if any point is either larger than upper threshold

(important feature) or larger than lower threshold and connected to a point that has a value larger than upper threshold (less important but connected to feature). An additional 2D texture, referred as the active-tag texture below, is used to record if any pixel on each slice is changed during the rendering. We downsample the active-tag texture, i.e. every four pixels are summed into a single pixel, until the texture shrinks down to a single pixel. Then glReadBuffers() reads in the final pixel value and stops if the value is 0. The pseudo code for the GPU hysteresis thresholding is given below (algorithm 1).

Algorithm 1 GPU Hysteresis Thresholding

```
upper ⇐ upper threshold
lower ⇐ lower threshold
run ⇐ TRUE
srcvol ⇐ input volume
tarvol ⇐ 0
while run is TRUE do
  activetag ⇐ 0
  for all slice s ∈ srcvol do
    for all pixel x ∈ s do
      if (x > upper) or (x > lower and any 1-neighbor is
      fault) then
        mark current pixel on S as fault
        mark current pixel on activetag as 1
      end if
    end for
  end for
  if any pixel in activetag is 1 then
    run = TRUE
  else
    run = FALSE
  end if
  swap srcvol and tarvol
end while
return srcvol
```

8. Results

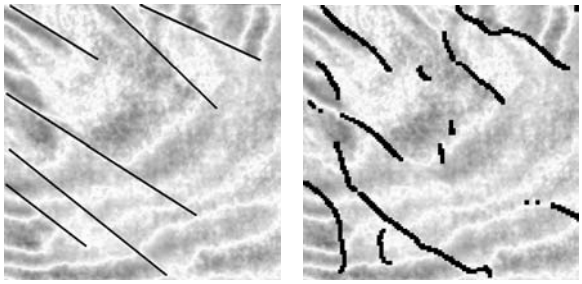
Here we describe an implementation of the GPU fault detection method on a PC equipped with a Pentium 4 3.6 GHz processor and an Nvidia 7800 GTX graphics card. We used Cg and glew library for the GPU programming and the openGL extensions. Figure 5 shows the screen shots of the several stages of the proposed system. Figure 5 (b) is the result of directional anisotropic diffusion, and (c) is the thresholded fault-likelihood volume. The center (or skeleton) of each blob in Fig 5 (c) has the strongest response of the fault-likelihood, and the response becomes weaker as moving further away from the center. Figure 5 (d) is the 3D fault voxels extracted from the fault-likelihood volume using a non-maximal-suppression and a hysteresis thresholding. Figure 5 (e) visualizes the faults on 2D slices.

Table 1 compares the running time of 10 iterations of the directional anisotropic diffusion, the most time consuming process in the system, on various sizes of datasets using a CPU and a GPU implementation. The first row in the table represents the x and y dimension of the input volume, where all the volumes used in our experiments consist of 128 z-slices. It takes around 1.5 seconds on the GPU for 10 diffusion iterations on an 128^3 volume, that is about 20-times speed up compared to the CPU implementation.

	64	128	256
CPU (Pentium 4 3.6 GHz)	7.5 sec	30 sec	2 min
GPU (Nvidia 7800 GTX)	0.3 sec	1.5 sec	5.5 sec

Table 1: *Running time comparison of 10 iterations of directional anisotropic diffusion on the CPU and the GPU*

In addition to the fast running time, the quality of the detection is also comparable to the manually selected faults. Fig 4 compares the resulting faults extracted by manually and using the proposed method. There are small false positives, but the proposed method picks up most of important faults accurately. Because our method runs at interactive rates, users can easily try various parameters to get the best result or choose connection faults sets from the 3D interface. The proposed method can produce 3D fault bodies automatically in a few seconds as opposed to the manual fault interpretation can be done only on 2D images. The extracted faults are 1-pixel thick voxels, so they can be easily converted into 2D polylines or 3D polysurfaces for other applications.

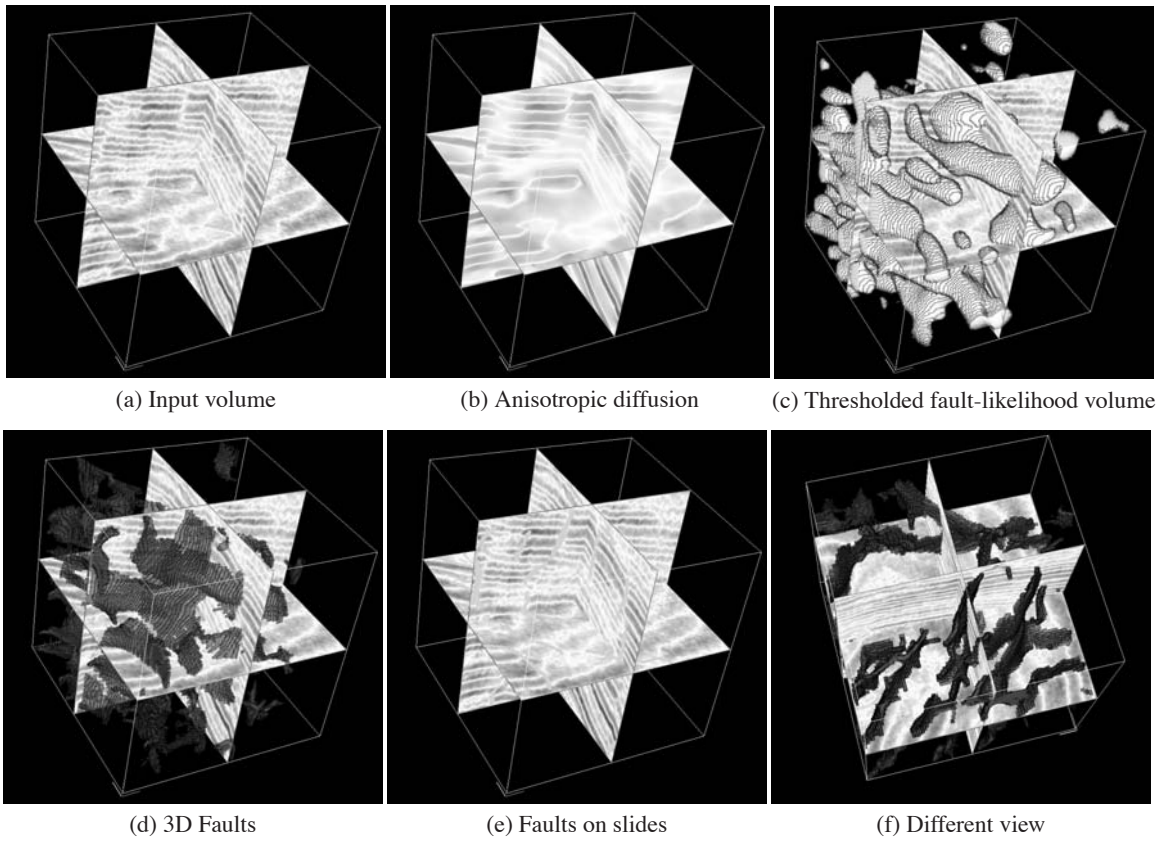


(a) Manually detected (b) Proposed method

Figure 4: *Fault detection comparison of manual and the proposed method.*

9. Conclusions

This paper describes a new seismic fault detection system for use with graphics processors. Local orientation of seismic structure is robustly calculated using structure tensor analysis, which is combined with anisotropic diffusion along seismic orientation. A fault-likelihood volume is computed using a directional variance measure, and 3D faults



(a) Input volume (b) Anisotropic diffusion (c) Thresholded fault-likelihood volume

(d) 3D Faults (e) Faults on slides (f) Different view

Figure 5: *Screen shots from each stage of the fault detection system*

are then extracted using non-maximal-suppression and hysteresis thresholding. The proposed system performs about 20-times faster than the CPU implementation in our experiments, that suggests promising future research directions for GPU applications. Future work includes developing new GPU computational models to deal with large seismic datasets, and developing methods to detect other seismic features, e.g. channels or salt domes.

References

[Bak02] BAKKER P.: *Image structure analysis for seismic interpretation.* PhD thesis, Technisch Universiteit Delft, 2002.

[BF95] BAHORICH M., FARMER S.: 3-D seismic discontinuity for faults and stratigraphic features : The coherence cube. *Leading Edge 14* (1995), 1053–1058.

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph. 22*, 3 (2003), 917–924.

[BGW91] BIGÜN J., GRANLUND G., WIKLUND J.: Multidimensional orientation estimation with applications to texture analysis and optical flow. *IEEE Transaction on Pattern Analysis and Machine Intelligence 13*, 8 (August 1991), 775–790.

[BvVV99] BAKKER P., VAN VLIET L., VERBEEK P.: Edge preserving orientation adaptive filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (June 1999), pp. 535–543.

[Can86] CANNY J.: A computational approach to edge detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence 8*, 6 (1986), 679–698.

[CC02] COHEN I., COIFMAN R. R.: Local discontinuity measures for 3-D seismic data. *Geophysics 67*, 6 (2002), 1933–1945.

[Cho02] CHOPRA S.: Coherence cube and beyond. *First Break 20.1* (January 2002), 27–33.

[FH03] FEHMERS G., HÖCKER C.: Fast structural interpretation with structure-oriented filtering. *Geophysics 68*, 4 (July-August 2003), 1286–1293.

[GM99] GERSZTENKORN A., MARFURT K. J.: Eigenstructure-based coherence computations as an

aid to 3-D structural and stratigraphic mapping. *Geophysics 64* (1999), 1468–1479.

[GST03] GIBSON D., SPANN M., TURNER J.: Automatic fault detection for 3d seismic data. In *Proc. VIIth Difital Image Computing: Techniques and Applications* (2003).

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *Proceedings of Graphics Hardware* (2003), pp. 102–111.

[HBPA01] HASAN K., BASSER P., PARKER D., ALEXANDER A.: Analytical computation of the eigenvalues and eigenvectors in dt-mri. *Journal of Magnetic Resonance 152* (2001), 41–47.

[HE99] HOPF M., ERTL T.: Accelerating 3d convolution using graphics hardware (case study). In *Proceedings IEEE Visualization 1999* (1999), pp. 471–474.

[HLB*05] HARRIS M., LUEBKE D., BUCK I., GOVINDARAJU N., KRÜGER J., LEFOHN A., PURCELL T., WOOLLEY C.: GPGPU:general purpose computation on graphics hardware. In *Course 39 at ACM SIGGRAPH* (2005).

[JM05] JACQUEMIN P., MALLET J.-L.: Automatic fault extraction using double hough transform. In *SEG Expanded Abstracts* (2005), vol. 24, pp. 755–758.

[KKKW05] KRÜGER J., KIPFER P., KONDRATIEVA P., WESTERMANN R.: A particle system for interactive visualization of 3d flows. *IEEE Transactions on Visualization and Computer Graphics 11*, 6 (11 2005).

[KKW05] KONDRATIEVA P., KRÜGER J., WESTERMANN R.: The application of gpu particle tracing to diffusion tensor field visualization. In *Proceedings IEEE Visualization 2005* (2005).

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: a GPU-based particle engine. In *Proceedings Graphics Hardware* (2004), pp. 115–122.

[KW03] KRÜGER J., WESTERMANN R.: Linear algebra operators for GPU implementation of numerical algorithms. In *Computer Graphics (SIGGRAPH '2003 Conf. Proc.)* (2003), pp. 908–916.

[LKHW03] LEFOHN A. E., KNISS J. M., HANSEN C. D., WHITAKER R. T.: Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings IEEE Visualization 2003 (VIS'03)* (2003), pp. 75–82.

[LLZ*05] LU W., LI Y., ZHANG S., XIAO H., LI Y.: Higher-order-statistics and supertrace-based coherence-estimation algorithm. *Geophysics 70*, 3 (2005), 13–18.

[MKFB98] MARFURT K. J., KIRLIN R., FARMER S. L., BAHORICH M. S.: 3-D seismic attributes using a semblance-based coherency algorithm. *Geophysics 63*, 4 (1998), 1150–1165.

[OLG*05] OWENS J., LUEBKE D., GOVINDARAJU N., HARRIS M., KRÜGER J., LEFOHN A., PURCELL T.: A survey of general-purpose computation on graphics hardware. In *State of the Art report, Eurographics 2005* (August 2005), pp. 21–51.

[PM90] PERONA P., MALIK J.: Scale-space and edge detection using anisotropic diffusion. *IEEE Transaction on Pattern Analysis and Machine Intelligence 12*, 7 (1990), 629–639.

[PRSS02] PEDERSEN S. I., RANDEN T., SONNELAND L., STEEN O.: Automatic fault extraction using artificial ants. In *SEG Expanded Abstracts* (2002), vol. 21, pp. 512–515.

[RPSS99] RANDEN T., PEDERSEN S. I., SIGNER C., SONNELAND L.: Image processing tools for geologic unconformity extraction. In *Proceedings of IEEE Nordic Signal Processing Symposium* (1999), pp. 9–11.

[RS01] RUMPF M., STRZODKA R.: Nonlinear diffusion in graphics hardware. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '01* (2001), pp. 75–84.

[SR01] STRZODKA R., RUMPF M.: Level set segmentation in graphics hardware. In *Proceedings of IEEE International Conference on Image Processing (ICIP'01)* (2001), 3, pp. 1103–1106.

[VKG03] VIOLA I., KANITSAR A., GRÖLLER M. E.: Hardware-based nonlinear filtering and segmentation using high-level shading languages. In *Proceedings IEEE Visualization 2003* (2003), pp. 309–316.

[Wei99] WEICKERT J.: Coherence-enhancing diffusion filtering. *International Journal of Computer Vision 31* (1999), 111–127.