Interactive GPU-based “Visulation” and Structure Analysis
of 3-D Implicit Surfaces for Seismic Interpretation

by

Benjamin James Kadlec

B.S., University of Minnesota, 2005
M.S., University of Colorado, 2006

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirement for the degree of

Doctor of Philosophy

Department of Computer Science

2009
This thesis entitled:

Interactive GPU-based “Visulation” and Structure Analysis of 3-D Implicit Surfaces for Seismic Interpretation

written by Benjamin James Kadlec
has been approved for the Department of Computer Science

__________________________
Henry M. Tufo (advisor)

__________________________
Geoffrey A. Dorn

Date: ___________

The final copy of this thesis has been examined by the signatories, and we Find that both the content and the form meet acceptable presentation standards Of scholarly work in the above mentioned discipline.
Three-dimensional seismic data has been used to explore the Earth’s crust for over 30 years, yet the imaging and subsequent identification of geologic features in the data remains a time consuming manual task. Current approaches fail to realistically model many 3-D geologic features and offer no integrated segmentation capabilities. In the image processing community, image structure analysis techniques have demonstrated encouraging results through filters that enhance feature structure using partial derivative information. These techniques are only beginning to be applied to the field of seismic interpretation and the information they generate remains to be explored for feature segmentation. Dynamic implicit surfaces, implemented with level set methods, have shown great potential in the computational sciences for applications such as modeling, simulation, and segmentation. Level set methods allow implicit handling of complex topologies deformed by operations where large changes can occur without destroying the level set representation. Many real-world objects can be represented as an implicit surface but further interpretation of those surfaces is often severely limited, such as the growth and segmentation of plane-like and high positive curvature features. In addition, the complexity of many evolving surfaces requires visual monitoring and user control in order to achieve preferred results.

This thesis presents a unified approach that combines image structure analysis and implicit surface modeling in an interactive “visulation” environment (IVE) specifically designed to segment geologic features. The IVE allows geoscientists to observe the evolution of surfaces and steer them toward features of interest using their domain knowledge. This work has been implemented on the GPU for increased performance and interaction. The resulting system is a
surface-driven solution for the interpretation of 3-D seismic data, in particular for the segmentation and modeling of faults, channels, and other geobodies.
Acknowledgements

I am extremely grateful to have spent the last 4 years as a graduate student in the computer science department at the University of Colorado at Boulder. The beautiful surroundings of Boulder and the wonderful people I have worked with during my time here as a graduate student have bred a well-balanced lifestyle that has been an ideal environment for conducting collaborative research. My research has allowed me to work with amazing people in a number of institutions in Boulder and around the world. All of these people have been amazing influences on me who have contributed in one way or another to both my career and my life.

I would first like the thank my advisor Henry M. Tufo, for whom I give credit for recruiting me to the University of Colorado and being the constant figure in my graduate career by guiding me through the process of graduate school and my research. Henry’s approach to advising allowed for a unique blend of independence and regimen in the way he oversaw my work. His advise to me was always honest, sincere, and well thought-out such that I rarely questioned his opinion on matters related to my research or life in general. Henry was dedicated to me and ensured that I always had the resources necessary to conduct my research. He also used every opportunity possible to connect me with other influential people in academia and industry. Most importantly, Henry has brought me into his personal life in a way that elevates the student-advisor relationship to something closer to friendship. It is also worth noting that my skiing skills have progressed incredibly thanks to Henry’s encouragement and some of our best conversations took place off-campus in the mountains of Colorado.

Next, I would like to thank the members of my committee including Elizabeth Bradley, Clayton Lewis, Jane Mulligan and Paul Weimer. Geoffrey Dorn receives special thanks for being the day-to-day advisor of my work and introducing me to the field of 3-D seismic interpretation, which was the motivation for many of the techniques developed in my thesis. Geoff allowed me the freedom to dive into unexplored research areas and to approach the field of seismic
interpretation in a rather non-traditional way. Geoff’s industry knowledge was essential for teaching me how to solve real-world problems and to understand the importance of how a geoscientist sees things differently from a computer scientist. He has been my guide in the world of 3-D seismic interpretation and provided me with the opportunity to meet many of the foremost experts in the field.

I would also like to thank my undergraduate advisor David Yuen, who in the year 2004 loaned me a book on level set methods without knowing that years later it would become part of my PhD work (the book was only recently returned to Dave). I have worked with Dave continually since my sophomore year at the University of Minnesota, and he has certainly been the most influential person in my research career. Dave gets credit for introducing me to the fields of supercomputing, visualization, and numerical computation, and he did it all while I was just a wide-eyed 19-year-old undergrad. Dave has been dedicated to both my work and my life as an advisor, a colleague, and a friend. For this and for so much more, I am greatly indebted to him.

Much of the research I conducted for my thesis was supported and influenced by fellow graduate students and researchers at TerraSpark Geosciences (formerly, the University of Colorado BP Center for Visualization). I would like to thank Jon Marbach for his work in designing the software environment known as CASI, which was used as the development platform for much of my research work. I would also like to thank Stan Hammon, Jim Carlson, and Mick Coady for their equally important work on CASI as well as a number of other tools and techniques that in one form or another supported my thesis work. In addition, Stan Hammon’s friendship and knowledge of the geosciences had a special influence on my work thanks to our years of lively research discussions that took place in various offices, bars, and U-Haul trucks.

I would lastly like to thank my family. My parents Scott and Lisa gave me an amazing childhood filled with fun, laughter, love, and an appreciation for hard work and determination. They’ve made sacrifices to provide me with every opportunity to succeed and be happy, and I will never be able to thank them enough for the life they’ve given me. Next, I’d like to thank my
siblings Becca, Joey, and Andy who have always been my best friends that I never get to visit often enough. I would especially like to thank my late Grandpa Ben, who was a role model throughout my youth and taught me what it means to be a true North Dakotan in ways I will never forget. Most of all, I owe this work to my wife Sara who has been my biggest friend and supporter without whom I likely would have never been able to survive the final thralls of my dissertation. She has been the eternal joy in my life when after weeks filled with 18-hour workdays and sleepless nights; she was always there for me.
# Contents

ACKNOWLEDGEMENTS .................................................................................. V
CONTENTS .................................................................................................. VIII
LIST OF FIGURES AND TABLES .................................................................. IX
GLOSSARY .................................................................................................... XV

1. INTRODUCTION ...................................................................................... 1
  1.1. PROBLEM STATEMENT ................................................................. 1
  1.2. OVERVIEW .................................................................................. 5
  1.3. CONTRIBUTION AND RESULTS .................................................. 6
  1.4. THESIS PUBLICATIONS AND INTELLECTUAL PROPERTY ............. 7

2. TECHNICAL BACKGROUND AND RELATED WORK ............................. 8
  2.1. TRADITIONAL SEISMIC INTERPRETATION TECHNIQUES ................. 8
  2.2. GEOLOGIC FEATURES IN SEISMIC DATA .................................... 12
  2.3. 3-D IMAGE STRUCTURE ANALYSIS ............................................. 17
  2.4. LEVEL SET METHODS .................................................................. 22

3. STRUCTURE ANALYSIS OF SEISMIC DATA ...................................... 36
  3.1. DIFFUSIVE SMOOTHING ............................................................. 36
  3.2. CRITICAL POINTS ....................................................................... 38
  3.3. CONFIDENCE AND CURVATURE .................................................. 40
  3.4. DIRECT SEISMIC FAULT ENHANCEMENT .................................... 43
  3.5. FACIES FOR SEISMIC SEGMENTATION ...................................... 45

4. LEVEL SET TECHNIQUES FOR GEOLOGIC SURFACES ................... 48
  4.1. CONFIDENCE AND CURVATURE GUIDED LEVEL SETS ................... 48
  4.2. PLANAR LEVEL SETS ................................................................. 54
  4.3. MEDIAL-SURFACES ................................................................. 59
  4.4. EXTERNALLY GENERATED MOTION IN THE NORMAL DIRECTION .... 69
  4.5. MEDIAL SURFACE-GUIDED SHAPE EXAGGERATION .................... 72
  4.6. MULTIPLE REGION FLOW ............................................................ 84

5. INTERACTIVE “VISUALIZATION” ENVIRONMENT (IVE) ....................... 87
  5.1. LEVEL SET “VISUALIZATION” ON THE GPU ................................ 87
  5.2. SEEDING LEVEL SETS ............................................................... 114
  5.3. SURFACE Merging ................................................................. 118
  5.4. MODIFYING SURFACES AND INTERACTIVE STEERING ............ 124
  5.5. USABILITY AND VALIDATION ..................................................... 127

6. CONCLUSIONS ....................................................................................... 155
  6.1. SUMMARY .................................................................................. 155
  6.2. FUTURE WORK ........................................................................... 155

BIBLIOGRAPHY ......................................................................................... 158

APPENDIX A: RESEARCH PLAN .................................................................. 163
APPENDIX B: NUMERICAL DISCRETIZATIONS ......................................... 164
APPENDIX C: SOFTWARE DESIGN .......................................................... 176
APPENDIX D: HUMAN STUDIES .............................................................. 179
CURRICULUM VITAE: BENJAMIN J. KADLEC ........................................... 182
List of Figures and Tables

Figure 1: Top box is an overview of the traditional seismic interpretation workflow and where the contribution of this thesis fits. Bottom box is a coarse pipeline of the technique proposed by this thesis. ................................................................. 4
Figure 2: 3D Seismic Dataset showing inlines, crosslines, and time directions. ................................. 9
Figure 3: The seismic strata (red-to-blue layering) are rarely perfectly horizontal. Green surface describes the correct local coordinate system for small section of the volume. ......................... 10
Figure 4: Wiggle traces overlain on a standard blue to red color-mapped seismic amplitude data. Strata can be observed as consistent positive (red, peak) or negative (blue, trough) amplitudes that can be connected horizontally across the data................................................................. 13
Figure 5: Seismic volume time slice (left, top) and inline (left, bottom) and corresponding stratal volume time slice (right, top) and inline (right, bottom). Strata are nearly flat in the stratal volume. ........................................................................................................ 14
Figure 6: Example of a swath of noise (outlined in blue) from a 2-D slice of a 3-D seismic dataset. The goal of noise filtering is to remove this noise while preserving feature boundaries. Before (top) and after (bottom) filtering is applied................................................................. 15
Figure 7: Simplified description of three types of faulting (left) and inline slice of a seismic dataset showing many crossing faults (right). ........................................................................................................ 16
Figure 8: (left) Inline/Crossline diagram of typical layering structure that forms channels and (right) a time-slice of a 3D volume where red arrows point to channel features. ............................ 17
Figure 9: Mean curvature flow shrinks high-curvature regions of an object to a single point (left to right, top to bottom)........................................................................................................ 27
Figure 10: 2-D example of the medial-curve for a simple rectangle shape. Left image shows the medial-curve in blue, while red points are the locus of the maximal disks given by dashed lines [9]. Right image shows the extracted medial-curve as segmented components differentiated by color. ........................................................................................................ 29
Figure 11: Histogram of values for a scalar volume (blue). Gray line represents an opacity curve, rainbow colors designate color map, min and max values set the clipping range of the histogram ........................................................................................................ 32
Figure 12: Generic framework describing many possibilities for level set surface editing. .................. 35
Figure 13: Slice of an original 3-D seismic image (left), statistically smoothed using a median filter (middle), and the stratigraphy-preserving anisotropically diffused image (right). Zooming in on the large channel shows the median filter shrunk the width of the channel (middle, bottom) while the stratigraphy-preserving anisotropic diffusion retained the original size width of the channel (right, bottom). In addition, other channel features visible in the original data as shown by the yellow and brown arrows begin to disappear when smoothed using median filtering (middle), yet they are preserved and enhanced in the anisotropically diffused data (right). ........................................................................................................ 38
Figure 14: (a): 1-Saddle, (b): 2-Saddle, and (c) Maximum critical points of a surface in 3D. Bottom image gives examples of each critical point type in a seismic fault dataset......................... 40
Figure 15: Channelness measure calculated in 3-D on the stratal slice shown on the left, with the resulting attribute on the right. Bright values correspond to a high likelihood of a channel. Channel edges can be found by computing the gradient of the attribute. ........................................... 43
Figure 16: Computation of vertically summed variance along the strata for fault imaging............. 45
Figure 17: Volume segmentation of a 1-D signal using the watershed algorithm. From left to right, watershedding is implemented by slowly raising a water level (blue) such that basins are submerged. When the water originating from two different basins comes together, a watershed is created to divide the regions (red dashed line). At the completion of the process, regions are automatically classified by the differentiating watersheds (colored blocks). ........................................................................................................ 46
Figure 18: Watershed segmentation of a seismic volume (left) into horizon intervals (middle) and fault “blocks” (right).

Figure 19: Computation of the inside of a channel by identifying high curvature on lateral slices (left) and location of channel edges based on the gradient on the boundary of a channel (right).

Figure 20: Visual representation of the contribution of level set terms in equation Equation 4-1 for evolving a surface (or contour) towards a bright intensity feature in 2-D.

Figure 21: Slice of channelness attribute of 3-D seismic volume overlaid by the red outline of the level set segmentation. Left to right, increasing iterations of 10, 50, 100 respectively.

Figure 22: Slice of channelness attribute of meandering channel from 3-D seismic volume, overlaid by the red outline of the level set segmentation. Left to right, increasing iterations of 10, 50, 100 respectively.

Figure 23: Three-dimensional representation of a segmented channel displayed in different orientations on a y,z-slice (top left), y- and z-slice (top right), x- and z-slice (bottom left), x- and z-slice rotated (bottom right).

Figure 24: Original seismic image on a z-slice different from Figure 11 (top left) and three-dimensional representation of the segmented channel displayed on the z-slice (top right), x- and on z-slice (bottom left), x- and z-slice rotated (bottom right).

Figure 25: In this figure two views of the bounding surface of a fault’s 2-D manifold are shown. Colored surfaces represent the actual 2-D fault manifold and silver surfaces are the bounding surface of the fault.

Figure 26: Threshold-based fault velocity functions for the sawtooth form.

Figure 27: Example of high-propagation evolution for (left) initial and (right) final time steps. Background grayscale image is the fault-likelihood data overlaid in red by the level set fault extraction. Black arrows points to initial seeds that shrunk and yellow arrow points to a new fault region that the technique discovered.

Figure 28: Comparing propagation only flow (left) to propagation flow with curvature flow (right) for the initial seeds (top). Blue represents the level set surface and red is the boundary of the surface. Bright features in the background image are faults and dark features are non-faults.

Figure 29: Fault surface evolving in 3-D from initial seed lineaments (left to right), for 50 iterations, 100 iterations, then the segmented medial-surfaces of the fault surface (right), as will be described in section 4.3.

Figure 30: Removal of non-medial surface critical points from the distance transform. Non-minimal suppression thinning is conducted using the gradient structure tensor to preserve the medial surface. (Top): Before non-minimal suppression and (Bottom): after non-minimal suppression, which results in a thin representation of the surface.

Figure 31: Connectedness search along the medial surface orientation as defined by the 3D structure tensor. Shortest path is a function of Euclidean distance and gradient magnitude. Top is before and bottom is after searching on a synthetic fault dataset.

Figure 32: Segmentation of the medial surface on a synthetic fault dataset. Colored surfaces represent merged components, circles represent the critical point under consideration, and arrows describe eigenstructure.

Figure 33: Medial-surface extraction and segmentation showing (a): original ‘hole cube’ shape, (b): distance transform, (c): medial surface on 2D slices, (d) maximum components, (e): 1-saddle components, (f): 2-saddle components.

Figure 34: Medial-surface extraction and segmentation showing (a): original ‘flat snake’ shape, (b): rotated original, (c): distance transform volume, (d): medial surface on 2-D slices, (e): rotated medial surface with different 2-D slices, (f): maximum components (very small), (g): 1-saddle components, (h): 2-saddle components, (i): rotated 1-saddle, 2-saddle, and maximum components combined.
Figure 35: Medial-surface extraction and segmentation results from two different seismic datasets. Top row shows Seismic-A and bottom row shows Seismic-B as (a,c,e): original level set simulation output, (b,f): level set distance transform, (c,g): medial surface slices, and (d,h): segmented components. ........................................................................................................... 68

Figure 36: Analysis of results on Seismic-A fault dataset as (left): 1-saddle and 2-saddle medial surfaces and (right) a maximum medial surface. Arrows approximate the local eigenstructure at the critical points. .............................................................................................................................. 69

Figure 37: Box growth (top to bottom, left to right) along the x-axis by external force <+/−1, 0, 0> .................................................................................................................................................. 70

Figure 38: Sphere-like growth (top to bottom, left to right) in the radial direction by external force <+/−1, +/-1, +/-1> ........................................................................................................................................... 70

Figure 39: Example of smoothing (blue) and exaggeration (red) of a simple 1-D curve (black). ........... 73

Figure 40: Example of evolution becoming unsteady for inverse curvature flow. ............................... 76

Figure 41: Approximating the curvature at each point (\(\phi=l\)) in the level set domain to the curvature of its closest point on the initial front (\(\phi=0\)). Solid black curve represents the zero-valued level set and dashed black curves represent the level set at different distances. Since the radius of curvature (red circles) is linearly related to the distance of level set values (\(\phi=dr\)), the curvature at the initial front can be assigned to nearest points in the domain (orange arrows). Radius of curvature is \(r\) for \(\phi=0\) and \(r+\phi\) for \(\phi \neq 0\). ...................................................... 78

Figure 42: Description of medial-surface anisotropic flow for a 2-D star, such that shapes deform and extend in the direction of their local medial surface (solid colored arrows). .......................................................... 79

Figure 43: Orientation-guided steering of a shape (red-dotted line). The normal of the shape’s orientation (double-headed “normal” arrow) is compared to a steering wheel of possible orientations. When the shape’s medial normal is perpendicular to the steering direction (i.e., orientations are parallel), maximum growth of the shape occurs along the steering direction. Conversely, when the angle between the medial normal and the steering direction is zero, growth of the shape along the steering direction is minimal. ............................................................ 81

Figure 44: Medial-surface guided curvature flow for 3-D cross object. Left to right, top to bottom, medial surfaces of object followed by iterations 0, 10, 20, 30, and 40 respectively. ........... 82

Figure 45: Medial-surface guided curvature flow for 3D linear object. Left to right, top to bottom, medial surfaces of object followed by iterations 0, 10, 20, 30, 40 respectively. ........... 82

Figure 46: Medial-surface guided curvature flow for 3-D gear object. Left to right, top to bottom, medial surfaces of object followed by iterations 0, 10, 20, 30, and 40 respectively. ........... 83

Figure 47: Medial-surface guided curvature flow for 3-D geologic fault dataset. Left to right, top to bottom, medial surfaces of faults followed by iterations of 0, 10, 20, 30, and 40 respectively. .......................................................................................................................... 84

Figure 48: Nvidia GeForce 295 GTX has a theoretical peak processing power of 1788 GFlop/sec. A GPU is installed to a PCI Express slot on a workstation. ................................................................. 90

Figure 49: CUDA thread decomposition. 3-D volume (circled in green) represents the entire grid being computed. Blocks (yellow cube) process regions of the volume using parallel threads (orange square) that compute a value for every voxel in the volume. ................................................................. 94

Figure 50: CUDA memory decomposition. 3-D volumes (circled in green) are stored on the device’s global memory (DRAM) as a 3-D texture. Blocks (yellow) of the volume are pre-loaded into the multiprocessor’s fast shared-memory before being computed. Each voxel of the volume is computed by a thread (ALU) in the multiprocessor and has on-chip access to data in the shared memory. ........................................................................................................... 95

Figure 51: Extracting the triangulated surface from a voxel representation of the level set surface on the GPU and rendering it on the GPU. ....................................................................................... 98

Figure 52: Tri-linear texture filtering on a seismic volume (top) and a level set volume (bottom). Left image is non-filtered and right image is filtered. ................................................................. 100
Figure 53: Time series computed on the GPU (left to right, top to bottom) showing a channel surface evolving from a line of seed points. ................................................................. 102
Figure 54: Time series computed on the GPU (left to right, top to bottom) showing a fault surface evolving from a seed point in a seismic dataset................................................................. 103
Figure 57: Percentage of compute load and bottlenecks of the GPU implementation determined by using timers for the compute load in order to determine time spent in each function and by varying the memory and core bus speeds in order to identify hardware bottlenecks. ... 106
Figure 58: Fixed-size bounding box interactive subvoluming of a large seismic survey. .......... 108
Figure 59: Segmentation of high-amplitude geobody in a 3-D seismic volume showing (a) user defined seed point to start evolution. (b) and (c) show the extracted isosurface of the level set while it evolves at 50 and 200 iterations respectively ......................................................... 109
Figure 60: Calculation of the structure tensor on the seismic data around a narrow-band of the level set returns propagation and advection terms on the fly for use in surface evolution. .. 110
Figure 61: Real-time structure analysis of a 3-D shape as it is segmented using level set evolution. The level set uses only a propagation term that is based on edges in the volume........... 112
Figure 62: Automatically extracted seed lineaments for seed points to the level set. Different colored lineaments represent distinct seeds that are approximated to align with faults in the data.................................................................................................................. 115
Figure 63: Example of Semi-Automatic refinement by using the output of an initial level set simulation (left) as the input to a second level set process for the purpose of filling in a gap in the segmentation (right). .................................................................................. 116
Figure 64: Manual Seeding of Level Sets for Planar Fault Extraction (white blocks represent manual seeds, green surface is the segmented fault). ................................................................. 116
Figure 65: Example of the complex situation that results when interpreting thousands of discrete surfaces in 3-D. Surface patches become occluded and are difficult to discern........... 117
Figure 66: Manual merging of surface patches. White arrows describe the motion of a mouse cursor ........................................................................................................................................ 118
Figure 67: Example of Smart Merging by selecting a patch for consideration (left) then highlighting all patches that meet the distance, normal dot product, and coplanarity constraints. (right) Highlighted points are then automatically merged into a new feature.. 120
Figure 68: Example of Hide Merging by (left) selecting a patch for consideration then (middle) hiding all patches that do not meet the distance, normal dot product, and coplanarity constraints. (right) The user can then manually choose which patches to merge with the patches still left displaying .......................................................................................... 121
Figure 69: Graph showing the relationship between smart merging and hide merging, making the assumption that an unknown optimal merge exists in the set of all surfaces. Smart merging parameters attempt to automatically merge less than the optimal, in order to prevent selecting too many patches. Hide merging parameters lie on the other side of optimal by trying to display more patches than optimal, in order to give the user a wider choice of patches to merge .................................................................................................. 122
Figure 70: Example of surface breaking by selecting a green surface for processing (left) then breaking the surface into discrete patches with similar distance, normal dot product, and coplanarity arrangements as described by the different colors assigned to points (right). .. 123
Figure 71: Evolution of fault based on a manual seed, followed by merging and surface creation. ........................................................................................................................................ 124
Figure 72: Computational steering by interactively removing growth regions of the surface..... 125
Figure 73: Computational steering by interactively adding growth regions to the surface........ 126
Figure 74: Left to right, adding blue seed points to the edge of a surface then evolving it for 30 iterations. Result is an extended version of the implicit surface .................................................. 127
Figure 75: Example workflow of the interactive “visulation” system described in this thesis.... 128
Figure 76: Using the distance transform representation of different segmentations to create a combined segmentation that is also represented as a distance transform. The distance transform at level-zero represents the intersection of the segmentations. Using higher levels provides a consensus representation.......................... 136

Figure 77: Difference sample volume and template volume using the distance transform. The resulting zero crossing represents the discrimination between the two features, and higher levels of the combined distance transform create a consensus representation of the combined segmentations........................................ 137

Figure 78: Combination between multiple volumetric interpretations (1, 2, ...n) using the distance transform. ........................................................................................................ 139

Figure 79: (top to bottom) Expert segmentation of fault surface (left) compared to multi-user combined result (right) for fault features 1, 2, and 3 respectively........................................ 143

Figure 80: (top to bottom) Expert segmentation of geobody surface (left) compared to multi-user combined result (right) for geobody features 1, 2, and 3 respectively........................ 148

Figure 81: The software design process combines into a segmentation application that consists of a data window, process window (workflows and algorithms), parameter window, and 3-D render window. .............................................. 178

Figure 82: University of Colorado Human Studies Request for Review Form .................. 179

Figure 83: Consent Form ............................................................................................... 179

Figure 84: University of Colorado Human Research Committee Determinaton Letter .......... 180

Table 1: CSG Operators of And, Or, Difference implemented as Copy, Paste, Cut respectively on two implicit level set models A and B represented by distance volumes Va and Vb with positive inside and negative outside values. ................................................. 34

Table 2: The peak throughput (measured by floating-point operations) of programmable GPUs has increased considerably over multi-core CPUs in recent years [28]. Peak performance on the Nvidia GeForce GTX 295 is 1788.48 GFlop/sec, while a quad-core Intel i7 Extreme 965 has a significantly lower peak theoretical throughput of around 51 GFlop/sec. ................. 88

Table 3: GPU versus CPU runtime for three datasets segmenting a geobody (Figure 59), a fault (Figure 54), and a channel (Figure 53) for 200 iterations. Speedup is approximated by the CPU times divided by GPU time. Frames per second (FPS) numbers are based on the GPU times. Faces are the number of triangles (in thousands) rendered as an isosurface. GPU times include data transfer to/from the CPU......................................................... 103

Table 4: List of voice commands accepted by software...................................................... 130

Table 5: Parameters exposed to the participant during the fault segmentation user study........ 141

Table 6: Time taken to interpret three fault features. Participants B, E, F used the non-interactive approach, while participants A, C, D, G, H used the interactive approach. Manual is the hand-drawn interpretations of the faults without using the techniques presented in this thesis.............................................. 144

Table 7: Accuracy of user-study results compared to the manual hand-drawn interpretation for the three fault features. Participants B, E, F used the non-interactive approach, while participants A, C, D, G, H used the interactive approach. A score of 100% accuracy is the same as identically matching the hand-drawn (manual) interpretations of the faults.................. 145

Table 8: Reliability of the technique as determined by the standard deviation of participants results. This is computed as the average percent differences between participant segmentations with the mean of all participant segmentations. A value of 0% means participant segmentations......................................................................................... 145

Table 9: Parameters exposed to the participant during the geobody segmentation user study .... 146

Table 10: Time taken to interpret three geobody features. Participants B, E, F used the non-interactive approach while participants A, C, D, G, H used the interactive approach. Manual
is the hand-drawn interpretations of the geobodies without using the techniques presented in this thesis.  

Table 11: Accuracy of user-study results compared to the manual hand-drawn interpretation for the three geobody features. Participants B, E, F used the non-interactive approach while .

Table 12: Reliability of the technique determined by the standard deviation of participants results. This is computed as the average percent differences between participant segmentations with the mean of all participant segmentations. A value of 0% means participant segmentations were orthogonal to each other and a value of 100% means that all participants resulted in the exact same segmentation.

Table 13: Comparison of minutes required to complete segmentation for both interactive and non-interactive modes. It can be seen that participants working in interactive mode produced faster results on average.

Table 14: Differences in the validation between interactive and non-interactive techniques. It can be seen that segmentations computed using interactive techniques generally produced more accurate results.
Glossary

- **Acquisition**: the generation and recording of seismic data using a *source* to generate vibrations and a *receiver* to record the response.
- **Channel**: the physical confines of a river, slough or ocean strait consisting of a bed and banks.
- **Computational Steering**: interactive control over a computational process during execution.
- **CUDA**: Compute Unified Device Architecture.
- **Diapir**: a relatively mobile mass, such as a salt dome, that introduces into pre-existing strata.
- **DT**: DistANCE Transform.
- **Explicit Surface**: a surface defined by an explicit function. Examples include polygon meshes and unstructured grids.
- **Fault**: a planar fracture in rock in which the rock on one side of the fracture has moved with respect to the rock on the other side.
- **Fault Damage Zone**: complex tectonic deformation producing many fine-grained rocks that surround a fault plane.
- **Filtering**: smoothing of data to remove noise and preserve features.
- **Geobody**: connected set of voxels in a seismic volume with similar geologic characteristics.
- **GST**: Gradient Structure Tensor.
- **GPU**: Graphics Processing Unit.
- **GPGPU**: General Processing on the Graphics Processing Unit.
- **Horizon**: geologic strata imaged in seismic data.
- **Implicit Surface**: a surface defined by an implicit function, also called a level set.
- **Interaction**: the cause and effect relationship between a user and software.
- **Interactive**: implies an active cause and effect relationship between a user and software.
- **Interpretation (Seismic)**: analysis of data to generate reasonable models and predictions about the properties and structures of the Earth’s subsurface.
- **IVE**: Interactive “Visulation” Environment.
- **Level Set Methods**: a numerical technique for tracking interfaces and shapes.
- **Medial-Surface**: a thin version of a 3-D shape that is equidistant to the shape’s boundaries. Also called a 3-D skeleton.
- **MRI**: Massive Resonance Imaging.
- **Non-Interactive**: implies an inactive cause and effect relationship between a user and software.
- **Play**: industry term that refers to the exploration and extraction of oil reserves from a specific region.
- **Segmentation**: partitioning an image into multiple regions.
- **Seismic Data (3D)**: set of numerous closely spaced seismic lines that provide a high spatially sampled measure of the Earth’s subsurface reflectivity.
- **Seismic Facies**: a group of seismic amplitude variations with characteristics that distinctly differ from those with other distinct characteristics.
- **Skeleton**: a thin version of a shape that is equidistant to the shape’s boundaries. Also called medial-axis.
- **Stratum (Strata)**: a layer of rock or soil with internally consistent characteristics that distinguishes it from contiguous layers.
• **Structure Tensor**: matrix representation of partial derivatives. The gradient structure tensor is defined by first-order partial derivatives.
• **Structure Analysis**: the set of physical laws and mathematics required to study and predict the behavior of structures.
• **Visulation**: word blend of “visualization” and “simulation” describing a coupled system where visualization and simulation occur concurrently.
• **Volume**: three-dimensional data stored in a dense format with rectangular dimensions. Also called a 3-D image.
• **Wiggle Trace**: display that shows seismic trace amplitude as an oscillating line about a null point.
1. Introduction

1.1. Problem Statement

Despite volatile economic conditions, long-term trends suggest that worldwide demand for energy will continue to grow in order to support continued world industrialization and improvement of living standards [36]. Because an efficient and cost effective global infrastructure for production and distribution of oil and gas already exists, it is expected to remain a convenient and economical source of energy for the foreseeable future [29]. Exploration for oil and gas begins with the collection of data from the Earth’s sub-surface and analyzing it for potential reservoirs and geologic features important to drilling and production. Unfortunately, most of the easy oil fields in the world have already been discovered and therefore current exploration efforts focus on difficult to reach fields or missed plays in already developed fields.

Analyzing the seismic data used to locate reservoirs is a complex task due to the data’s unique layered structure, the difficulty in identifying features, the high level of noise, and the large size of data sets. In addition, increased acquisition has created an explosion in the amount of seismic data that needs to be analyzed; yet the oil industry is experiencing a drastic shortage of interpreters as the current generation nears retirement [32]. This presents a great opportunity for computer-aided techniques to be developed in order to aid the geoscientist in recognizing features in seismic datasets.

A similar problem exists in the medical imaging community for analyzing CT and MRI scans of patients as well as data generated by the visible human project [134]. Research in this area has developed many fundamental techniques in the area of image structure analysis and surface segmentation for 3-D volumetric data [60]. Although there are many corollaries between the features represented in medical and seismic datasets (e.g. depositional channel features [65] have a similar character to vascular systems [41]), it is not straightforward to apply the techniques
developed for medical imaging due to vastly differing noise character, local orientations, and the structure of features specific to seismic datasets.

Image structure analysis techniques enhance feature structure using partial derivative information. This approach is powerful because it employs a combination of first and second order derivative information to differentiate between a wide variety of structures. These techniques are only beginning to be applied to the field of seismic interpretation and the information they generate remains to be explored for feature segmentation. This presents an opportunity to adapt image structure analysis in order to create representations for geologic features that can be used to allow for easier segmentation.

Surface segmentation separates features from background data using either an implicit or explicit representation of a surface. Up until recently, most published work in computer graphics and vision for imaging applications have used explicit surfaces constructed from triangles. Deforming triangulated surfaces require extreme care to be taken when discontinuous topological changes occur and smoothness is difficult to guarantee. In addition, there is no guarantee that the result of a deformed explicit surface will be physically realizable. Implicit surfaces are represented volumetrically using level set methods [97] and have an advantage over explicit surfaces in how easily dynamic topological changes and geometric quantities, such as normals and curvatures, are computed. Also, the results of level set simulations are physically realizable implicit surface models, which is desirable when attempting to represent geologic features. The challenge remains in devising level set methods to represent the evolution of unique geologic shapes such as planar 2D manifolds and high curvature features, as well as determining how to adapt the surface to follow features imaged in seismic data.

Three-dimensional seismic data interpretation is a challenge in both imaging and segmentation where the ultimate goal is to automatically segment all features contained in a data set. Unfortunately, the science of geology has many unknowns and the seismic data used to represent it requires a trained eye and subjective analysis that cannot be reliably automated.
Similar problems are manifested in other imaging fields such as when an evolving surface segmentation requires human knowledge to proceed, possibly due to poor imaging or a highly complex feature. In order to account for unknowns in data, visual monitoring and user control of the segmentation that employs domain knowledge is necessary; this is a non-trivial exercise. Therefore, a need and opportunity exists to incorporate image structure analysis and implicit surface modeling into an interactive environment for segmentation. This thesis presents a unified approach in the form of an Interactive “Visulation” (simultaneous visualization and simulation) Environment (IVE) designed to efficiently segment geologic features with high accuracy. The IVE unifies image structure analysis and implicit surface modeling as a surface-driven solution that assists geoscientists in the segmentation and modeling of faults, channels, and other geobodies in 3-D seismic data.
Figure 1: Top box is an overview of the traditional seismic interpretation workflow and where the contribution of this thesis fits. Bottom box is a coarse pipeline of the technique proposed by this thesis.
1.2. **Overview**

Chapter 2 begins with an introduction to 3-D seismic data and traditional techniques used for interpretation. Then, specific geologic features found in seismic data are described. Next, previous work and background information is given for three-dimensional image structure analysis and level set methods. Chapter 3 describes the details on image structure analysis applied to seismic datasets. The first section introduces a diffusive smoothing filter for enhancing stratigraphic features, while following sections use first and second order structure tensors to identify critical points, image curvature, and other seismic data properties. Chapter 4 presents level set methods for segmenting geologic surfaces such as channels and fault influence zones as well as techniques for medial-surfaces and enabling surface motion according to geologic constraints. Chapter 5 presents the Interactive “Visulation” Environment (IVE) consisting of a streaming GPU implementation of structure analysis and level set techniques, interactive steering and segmentation capabilities. The usability of the IVE is then discussed followed by the description and results of a user study examining two seismic interpretation tasks and the value of interaction. Finally, chapter 6 provides a summary of the work and suggests future research directions.
1.3. **Contribution and Results**

This thesis makes significant contributions to computer science in the areas of visualization, numerical computation, pattern recognition, computer vision, and high performance computing. Due to the interdisciplinary nature of this thesis, additional contributions have been made in the geological sciences in the area of seismic interpretation. Contributions are summarized below:

**Computer Science**

- Structural Smoothing of 3-D Datasets
- Curvature and Confidence Features in 3-D Datasets
- Computation of 2-D Planar Level Sets in 3-D
- Level Set Segmentation guided by Image Structure
- Medial-Surface Segmentation of 3-D Shapes
- Inverse Curvature Flow and Exaggeration of Implicit Surfaces
- Multiple-Attribute Flow using Level Sets
- Interactive Steering of Level Set Surface Evolution
- Streaming GPU Level Set Algorithm and Implementation using CUDA
- Distance Transform Evaluation of Implicit Surface Models
- User Study on Interactive Techniques and the Value of Interaction

**Geological Science**

- Depositional Channel Segmentation Workflow using Structure Analysis and Level Sets
- Geologic Fault Extraction Workflow using Structure Analysis and Level Sets
- Modeling Fault Zones using Level Sets
- Geobody Segmentation Workflow using Structure Analysis and Level Sets

Section 1.4 provides a list of publications resulting from the research conducted for this thesis.
1.4. **Thesis Publications and Intellectual Property**

**Journal Papers** (1 published, 2 in-progress)

**Peer Reviewed Conference Proceedings** (4 published, 1 in-progress)

**Non-Peer Reviewed Conference Proceedings** (2 published)

**Symposia**

**United States Patent Applications**
- Fault Segmentation in 3-D Seismic Data using Implicit Surfaces (*provisional 2009*)
- Channel Segmentation in 3-D Seismic Data using Implicit Surfaces (*provisional 2009*)
- Structure Tensor Analysis of 3-D Seismic Data (*provisional 2009*)

**Industry Exposure**
- 2008 GIVC (Geoscience Interpretation and Visualization Consortia) Meeting.
- 2007 GIVC (Geoscience Interpretation and Visualization Consortia) Meeting.
- 2006 GIVC (Geoscience Interpretation and Visualization Consortia) Meeting.

Research results presented at three annual meetings attended by geoscientists and computer scientists from BHP Billiton, BP, Chevron, Conoco Phillips, Paradigm, Shell, and others.
2. Technical Background and Related Work

2.1. Traditional Seismic Interpretation Techniques

Seismic data for exploration and development is created by a controlled source that generates an elastic wave field, which moves downward into the earth and bounces back to the surface according to the character of rock at different depths. The first seismic datasets were collected along two-dimensional sections of the earth. Although significant success was achieved using 2-D datasets, interpretations based on them often lacked the detail necessary to reliably develop complex plays [35]. The transition from 2-D to 3-D seismic surveys as the dominant data used in exploration occurred in the late 1980s and early 1990s as new computer technologies made interpretation of 3-D datasets tractable. The first three-dimensional seismic survey was shot by Exxon near Houston, Texas in 1967 [14], the concept of a three-dimensional survey was later formalized by Walton [135] in 1972, and in 1976 the first results of the technology were publicly presented by Bone, Giles, and Tegland [11]. Three-dimensional surveys promised to improve the quality of interpretation in order to minimize uncertainties in the data so that risks in exploration, development, and production could be reduced (see [35], [14], and [139] for more information). A majority of the interpreters in the industry trained and practiced on the interpretation of 2-D data for a large part of their careers, and the advent of 3-D interpretation posed new conceptual challenges for working in three-dimensions.

The first 3-D data volumes were interpreted manually on paper plots that represented the seismic waveforms as wiggle traces. This technology converted to displaying horizontal slices of the volume using a filmstrip, as in a motion picture. The advent of computer workstations with color screens greatly improved upon these paper-based analog techniques by allowing geoscientists to digitally look at 2-D slices of the volume. The restriction to looking at 3-D seismic data in terms of 2-D slices comes from technological restrictions (computers could only
visualize 2-D) and the rectangular shape of the data as shown in Figure 2. The advent of 3-D computer graphics made analyzing datasets easier by allowing visualization using three-dimensional surfaces and volume rendering. Unfortunately, few algorithms exist to do true 3-D interpretation on the entire data volume and most are simply 2-D techniques applied to multiple slices.

Figure 2: 3D Seismic Dataset showing inlines, crosslines, and time directions.

2.1.1. Seismic Data Details

The three coordinates of a 3-D volume are commonly called *inline* (x), *crossline* (y), and *time or depth* (z). The inline direction coincides with the direction of boat movement or cable layout while acquiring data and the crossline is the direction perpendicular to this in the horizontal plane. The time or depth direction is the vertical axis through the volume that coincides with a depth below the surface of the earth or the two-way travel time for the seismic wave to travel from the Earth’s surface down to a reflecting boundary at a particular depth and back up to the surface. Due to the orthogonal design of seismic datasets (i.e., rectangular grids), interpreters typically visualize and analyze the data in a fashion similar to what was done two-dimensionally. 3-D datasets are typically visualized by displaying 2-D textures of inline, crossline, or time slices through the volume (Figure 3). It is important to notice that these pre-defined directions generally do not align with the underlying geology of the survey (such that features may not be imaged on
inline, crossline, or time slices). For instance, features may appear diagonally between an inline and crossline obliquely (such as along the x-y axis) in which case neither inlines nor crosslines will image the feature well. This is a problem with the way 3-D data is interpreted since the algorithms typically used are perpendicular to one of the directions and operate under the planar restriction of an inline, crossline, or time slice. An additional problem with traditional interpretation is that a time slice is often algorithmically mistaken to align with the strata, or horizontal layering, of seismic data, which is rarely perfectly horizontal and normally changes throughout a dataset (Figure 3). An example of this is the famous coherence cube algorithm [5].

![Seismic strata and coherence cube](image)

Figure 3: The seismic strata (red-to-blue layering) are rarely perfectly horizontal. Green surface describes the correct local coordinate system for small section of the volume.

After a seismic dataset has been acquired in the field and processed into a 3-D volume, a three-step approach is typically taken to interpret the data. First the data is filtered to remove random and coherent noise as well as to enhance certain features. Next structural features resulting from deformations and distributions of rock layers are interpreted. Lastly, stratigraphic features resulting from rock layering and sedimentary processes are extracted [34]. This high-level workflow (see Figure 1) is typically an iterative process where the results of one step are used to influence interpretation in an earlier or later step and the interpretation is continually
refined until an acceptable result is achieved. Geologic features are described in more detail in section 2.2.

### 2.1.2. Existing Commercial Technologies

A number of commercial Seismic Interpretation Systems are available to geoscientists such as VoxelGeo®, Petrel®, GeoProbe®, and Kingdom Suite®. These packages each have components for interpreting various geologic features, but the work presented in this thesis describes a system for interpretation that does not currently exist in industrial software or published research. In particular, it is a novel idea to automatically steer surfaces according to seismic data structure in order to represent geologic features.

A wide variety of seismic interpretation techniques have been implemented in the major commercial interpretation systems. Generally, these techniques involve manual interpretation of geologic features imaged in the seismic volume or an attribute volume created by applying various mathematical algorithms to the seismic volumes as a batch process. Visualization is then typically only available as a post-processing step. This two-stage interpretation process is problematic since it introduces a memory lapse in the time between setting parameters and viewing results. This removes geoscientists from using their geologic insight to its fullest extent since efforts are focused on understanding the techniques used to interpret data rather than actually focusing on features in the data. In addition, these systems require multiple intermediate attribute volumes to be calculated that must be inspected over a sequence of steps before a final interpretation is created. These commercial techniques offer little real-time interaction and provide limited editing tools. For example, the AFE fault extraction technique in VoxelGeo® creates fault surfaces that have noisy boundaries, sparse representations, and only provides grouping of objects based on global strike directions. The Ant Tracking fault extraction technique in Petrel also creates noisy fault surfaces that are not constrained by topological parameters and
are difficult to interact with or group together. Moreover, none of these fault extraction techniques attempt to model the influence zone around a fault.

Aside from a lack of real-time interaction and visual interpretation, there are also issues with how commercial systems go about modeling geologic features. The majority of these commercial systems process a seismic volume voxel by voxel in a greedy fashion by classifying voxels as features, possible-features, or non-features. This is a straightforward way to process volumetric datasets and can generate good results, but it is based solely on voxel properties and does not take into account the topology of a feature or treat it as a solitary object. By instead representing geologic features as implicit surfaces, they can be treated as discrete objects capable of a wide variety of deformations, topological constraints, and can be easily visualized.

2.2. Geologic Features in Seismic Data

This section presents a high-level overview of the geologic features that will be addressed in this thesis. The descriptions are written such that a layperson with minimal experience in the geosciences is able to understand the goal at hand. Particular types of structural and stratigraphic features are discussed, as well as the noise character of seismic data. These geologic features were chosen because they are the ones most commonly looked for when interpreting seismic data.
Figure 4: Wiggle traces overlain on a standard blue to red color-mapped seismic amplitude data. Strata can be observed as consistent positive (red, peak) or negative (blue, trough) amplitudes that can be connected horizontally across the data.

### 2.2.1. Seismic Amplitude

The fundamentals of seismic interpretation are based on interpreting changes in the amplitude signal of seismic traces. Since large impedance contrasts at geologic boundaries will generally have higher amplitudes in a seismic trace, changes in the seismic amplitudes coincide with changes in geology [52]. Seismic data was first visualized using wiggle traces, which was proceeded by using color-mapped amplitude slices (Figure 4). Color-mapped amplitude data is traditionally visualized using a blue-red palette such that for 8-bit resolution negative amplitudes are in the blue spectrum, positive amplitudes are in the red spectrum, and zero amplitude is white. *Geobodies* are sets of connected voxels in seismic data that have the same impedance characteristics and therefore similar amplitudes.
A geologic horizon is a reflection from a specific layer or stratum of rock with homogeneous composition. For the purposes of seismic interpretation, a horizon is therefore defined to be a laterally extensive surface occurring at a particular amplitude feature and extending across neighboring traces. As mentioned in section 2.1.1, the horizon is rarely horizontal and dips up and down as it moves through a dataset. A technique for representing a seismic volume such that all horizons are flat was presented in [48] for creating a so-called stratal volume, such that the effects of structural deformations (including faulting) are removed, and horizons are nearly flat (Figure 5). The stratal volume is used either as an alternative or to complement the standard seismic volume for interpreting features that exist along the orientation of the strata. These features can be easily visualized in the stratal volume by rendering a horizontal slice through the data, which is equivalent to an arbitrary slice that follows one particular horizon event through the seismic volume. This data representation is used throughout
this thesis for identifying geobodies based on amplitude and segmenting stratigraphic features (3.3).

### 2.2.2. Noise Characteristics of Seismic Data

![Figure 6: Example of a swath of noise (outlined in blue) from a 2-D slice of a 3-D seismic dataset. The goal of noise filtering is to remove this noise while preserving feature boundaries. Before (top) and after (bottom) filtering is applied.](image)

Seismic datasets always contain some level of noise from sources such as signal loss in the Earth (random noise) and acquisition footprint (coherent noise), which is a pattern associated with the way data was collected. Since noisy data seriously degrades the quality of interpretation results, it is necessary to filter datasets in a manner that preserves edges and feature boundaries. Figure 6 gives an example of a time slice of a seismic volume with a swath of random noise. It is important to note that this noise appears on multiple slices in 3-D, although only a single slice is shown here. When discussing noise removal in this thesis, the goal is to suppress the random noise in the volume, which provides enhanced imaging of features that extend across these noisy regions. Although, noise removal always comes with the tradeoff that data containing no noise
may unintentionally have smoothing applied to it, which can result in the loss of important
details. An improper smoothing techniques may also do a poor job of removing noise and simply
signal quality. Section 2.3.1 gives an overview of common noise filtering and smoothing
techniques.

![Image](image.png)

Figure 7: Simplified description of three types of faulting (left) and inline slice of a seismic
dataset showing many crossing faults (right).

### 2.2.3. Structural Features

*Faults* are the most common structural features interpreted in seismic data. A fault is a
fracture in the Earth’s crust along which displacement has occurred. This displacement may have
both vertical and horizontal components. Faults are important for interpreting seismic data
because they provide clues to the relative location of depositional layers in a survey and act as
*traps* for oil and gas reservoirs. In particular, faults can be *sealing* and block fluid flow through
rocks, or conversely they can provide a path for fluid to flow when they are not sealing.
Therefore, geoscientists spend a significant amount of time mapping out faults, which can
number in the thousands for a moderate sized survey. Figure 7 (right) shows an inline slice with
multiple faults, which can be seen as linear discontinuous features moving vertically through the
slice. Viewing multiple slices at once can describe how the faults extend in 3-D. The goal of fault
extraction is to segment these fault discontinuities as a 2-D manifold in three-dimensions. An
additional goal is to determine the three-dimensional envelope around a fault plane that coincides
with the thickness of the damage zone caused in the region near the fault.
2.2.4. Stratigraphic Features

Stratigraphy studies the ancient layering or deposition of rock layers on top of each other. Features in seismic datasets represented in this way are called stratigraphic features. Channels are stratigraphic features having a sedimentary structure often formed by flowing water and have a distinctive morphology or shape. Figure 8 (left) shows the cross section of the three most common types of layering that form channels, although there are many other situations under which channels may be formed, all display some amount of curvature in their structure. This layering is reflected in the structure of the channel feature in a 3-D data volume. These channel features are most easily recognized by looking at a stratal slice that approximates a depositional surface at some time in the past, as can be seen in Figure 8 (right). The goal of stratigraphic segmentation is to extract the bounding surfaces of channels or other stratigraphic features (e.g., sand bars) having similar character.

2.3. 3-D Image Structure Analysis

The idea of structure analysis of a 3-D dataset comes from the field of image processing where the structure of a 2-D image is represented as gradients, edges, or similar types of information [87]. This translates to 3-D data where gradients, edges, curvature, and other image elements are employed three-dimensions. This information is gained by calculating derivatives.
and partial derivatives, and then analyzing the vector representation of magnitude changes of pixel (or voxel in 3-D) values. In two-dimensions the orientation of maximum change in an image corresponds to Equation 2-1, where $I_x$ is the partial derivative of image I in the x-direction, and $I_y$ is the partial derivative in the y-direction.

$$|g| = \sqrt{I_x^2 + I_y^2}$$

$$\theta = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$

Equation 2-1

$$I_x = \frac{\partial I}{\partial x}$$

$$I_y = \frac{\partial I}{\partial y}$$

Equation 2-2

The vector resulting from this is directed according to the ordering of pixel points and points along the orientation of the angle $\theta$, which varies from $[0, \pi)$ with a magnitude given by $g$. Another helpful way to consider this vector is to think of it as the normal vector to a gradient contour in the image, which is more intuitive when working with level sets (in section 2.4.1). The calculation of the $I_x, I_y$ partial derivatives (Equation 2-2) can be accomplished using standard central differences between neighboring pixels (voxels) or more robustly by convolving neighboring voxels with a Gaussian mask over a range of voxels and then taking the difference of the Gaussian-smoothed neighbors.

2.3.1. Filtering and Smoothing

Filtering techniques such as Gaussian convolution, mean filtering, and median filtering are typically used in practice for smoothing noisy seismic images. These techniques are tuned for filtering in a way that enhances fault structure [5], and oriented filtering methods have been developed to enhance fault structure while still honoring the seismic strata [6][23][61]. Most of these oriented filtering techniques are designed to enhance fault structure in a way that doesn’t
over-filter discontinuities [40][59] by limiting the amount of anisotropic filtering near fault zones [61].

Mean and median filtering suppress noisy data by calculating the mean or median voxel value over a user-defined region of the volume and then assign that value to the voxel under consideration. Median filtering is usually preferred since a mean filter will be more influenced by extreme values and uses averaging to assign a new value. The median filter will tend to eliminate these random noise spikes without having to average the data since different numeric values are not created and a value appearing somewhere in the region is guaranteed to be assigned. These filters can be applied either on time slices, inlines, crosslines, or three-dimensionally using a neighborhood operator consisting of multiple inlines, crosslines, and time slices (called a slab). Since these techniques typically have no knowledge of the orientation of the underlying seismic strata, they are prone to blurring across horizons (or strata) if too large an operator is used. This blurring across horizons can quickly destroy important details in the volume and is therefore an undesired result. In addition, if fault zones are not respected, smoothing can proceed across a fault such that the discontinuities describing it are no longer visible.

2.3.2. Gradient Structure Tensor

The orientation of seismic strata are generally not horizontal (parallel to the ground plane), which means filtering techniques used on seismic images must take into account local orientations [61], otherwise undesired blurring across horizons will inevitably result as in the case of mean and median filtering. To measure the orientation of seismic strata, the gradient structure tensor (GST) is used, which has been extensively studied in computer vision and image processing [7][40][59][61]. For a local neighborhood $I(x,y,z)$ in a 3-D image the GST is given by Equation 2-3.
As described in [61], since the GST represents an orientation rather than a direction, this formulation allows the blurring of tensors in a way that lets vectors pointing in opposite directions to support an orientation rather than counteract each other.

The GST is a 3x3 positive semi-definite matrix, which is invariant to Gaussian convolution. Therefore, Gaussian convolution can be used to average the tensors in a way that creates a more robust representation of the orientation field. The eigenanalysis of the GST provides information about the local orientation and coherence of the seismic data. Eigenvectors define a local coordinate axis while eigenvalues describe the local coherence, which represents the strength of the gradient along the respective eigenvectors. The dominant (i.e., strongest) eigenvector represents the direction of the gradient orthogonal to the seismic strata, while the smaller two eigenvectors form an orthogonal plane parallel to the seismic strata. Near faults or other discontinuities in the data, the strength of the dominant eigenvector before Gaussian smoothing is not sufficient to confidently define a plane orthogonal to the strata (Figure 3). However, after Gaussian smoothing of the tensors, a more confident eigenstructure is represented at faults and discontinuities that more accurately represent the true orientation. The orientation of the respective eigenvectors provides a robust estimate of the local orientation at each point in the image. This orientation may be described by two angles, the dip angle $\theta$ and the azimuth angle $\phi$ using the three components of the eigenvector $(e_x, e_y, e_z)$ as defined by

\[
\text{GST} = \begin{bmatrix}
I_x^2 & I_xI_y & I_xI_z \\
I_xI_y & I_y^2 & I_yI_z \\
I_xI_z & I_yI_z & I_z^2
\end{bmatrix}
\]
\[
\cos \phi = \frac{e_x}{\sqrt{e_x^2 + e_y^2}} \\
\sin \phi = \frac{e_y}{\sqrt{e_x^2 + e_y^2}} \quad \text{Equation 2-4} \\
\cos \theta = \frac{e_z}{\sqrt{e_x^2 + e_y^2 + e_z^2}}
\]

where \(0^\circ < \phi < 360^\circ\) and \(0^\circ < \theta < 180^\circ\).

### 2.3.3. Hessian Tensor

The Hessian is computed as the matrix of the second-order partial derivatives of the image (or volume). The Hessian tensor is given by

\[
H = \begin{bmatrix}
I_{xx} & I_{xy} & I_{xz} \\
I_{xy} & I_{yy} & I_{yz} \\
I_{xz} & I_{yz} & I_{zz}
\end{bmatrix} \quad \text{Equation 2-5}
\]

where second partial derivatives of the image \(I(x,y,z)\) are represented as \(I_{xx}, I_{yy}, I_{zz}\), and so forth. The eigenvalues of this tensor are ordered as \(\lambda_1 > \lambda_2 > \lambda_3\) and their corresponding eigenvectors as \(v_1, v_2, v_3\), respectively. Using the eigenvalues, this tensor can classify local second-order structures that are plane-like, line-like, and blob-like. [87] states the conditions for which the different eigenvalues describe these features as:

- **Blob-like**: \(\lambda_1 = \lambda_2 = \lambda_3\)
- **Plane-like**: \(\lambda_1 >> \lambda_2 = \lambda_3\)
- **Line-like**: \(\lambda_1 = \lambda_2 >> \lambda_3\)

By employing second-order information in the dataset, it is not possible to calculate curvature, corners, flatness, and other 2\textsuperscript{nd} order information. This analysis will be used for imaging
confidence and curvature features in section 3.3 and further applied using similar analysis for the use of locating critical points for medial surface extraction in section 3.2.

2.4. Level Set Methods

2.4.1. Level Set Equation

Level sets [97] are an implicit representation of a deformable surface. The use of level sets has been widely documented and comprehensive reviews on the method and associated numerical techniques can be found in [99] and [124]. The advantage of level set methods is that instead of manipulating a surface directly, it is embedded as the zero level set of a higher dimensional function called the level set function. The level set function is then evolved such that at any time the evolving surface can be implicitly obtained by extracting the zero level set.

An implicit representation of a surface consists of all points $S = \{i \mid \phi(i) = 0\}$, where $\phi: R^3 \Rightarrow R$. Level sets relate the motion of the surface $S$ to a PDE on the volume as

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \nabla \phi$$

where $V$ describes the motion of the surface in space and time. This equation is able to implement a number of deformations by simply defining an appropriate $V$. This velocity term can be combined with a variety of other terms such as image-dependent terms and geometric terms (e.g., mean-curvature) [82]. Equation 2-6 is sometimes referred to as the level set equation, as it was introduced by Osher and Sethian [97].

The initial level set must be represented as a signed distance function where each level set is given by its distance from the zero level set. The distance function is signed so there is differentiation between the inside and outside of the level set surface. For this work all points contained within the level set surface are considered to be negative distances. The distance function is computed using a technique that solves the Eikonal equation [103], which is
commonly done using the fast marching method [123] or the fast sweeping method [148]. This equates to a surface expanding in the normal direction with unit speed and can be considered a special case of the level set function.

The surface integral (surface area) and the volume integral of the surface $S$ can be defined using the implicit representation of the level set. The Dirac delta function on the interface is defined as

$$\delta(i) |\nabla \phi|$$

Equation 2-7

and the Heaviside function (integral of the Dirac delta function) as

$$H(i) = \begin{cases} 1 & \text{if } \phi(i) \geq 0 \\ 0 & \text{if } \phi(i) < 0. \end{cases}$$

Equation 2-8

Using these functions one can derive the surface area integral (in 3-D)

$$\int_S \delta(i) |\nabla \phi| di$$

Equation 2-9

and the volume integral

$$\int_S H(-i) di .$$

Equation 2-10

Additional intrinsic geometric properties of the implicit surface can be determined using this formulation. For instance, the normal is computed on the level set as

$$\vec{n} = \frac{\nabla \phi}{|\nabla \phi|}$$

Equation 2-11

and the curvature is obtained as the divergence of the normal as

$$\kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} .$$

Equation 2-12

Details of the computation of these terms are described in the Appendix B.7.


2.4.2. Efficient Computation

The level set can be solved in the naïve fashion by updating the entire domain every iteration, which results in time and space complexity of $O(n^3)$ for a volume of size $n^3$. Since the only important result is the computation at the zero level set ($\phi=0$), it might appear at first that a restriction can be made to updating the solution only at points adjacent to the moving front. However, in order to maintain a consistent distribution of level sets–per-volume, care must be taken to ensure the level set density does not become too spread out (low density) or too close together (high density). Too low of a level-set density makes it difficult to locate and isolate computation on the zero level set, while too high of a density can cause numerical problems as the calculation of derivatives becomes unstable. These problems are alleviated by maintaining a consistent signed-distance transform across the level set by frequently re-calculating the signed distance transform using the fast marching method or an equivalent technique.

The narrow-band technique [1] establishes a limited region around the zero level set where computation occurs, while the remainder of the domain remains stored in memory and is not computed at each iteration. The narrow-band, which is computed every iteration, is typically on the order of 5-15 voxels around the zero level set and reduces the time complexity to $O(n^2)$, although the space requirement of $O(n^3)$ is unchanged. A similar technique, also $O(n^2)$ in time, called the sparse-field approach [142] maintains a linked list of active voxels around the zero level set and only updates the active field. These efficient implementations of the level set method require the band (or sparse field) to be reinitialized whenever the evolving front approaches the border with the remaining computational domain or when the level set density becomes too dense or spread out. This is accomplished by re-computing the distance transform, a somewhat expensive process ($O(n^2 \log n^2)$ or $O(n^3)$ with a large constant).

Even with these more efficient implementations where only a narrow-band of the level set is computed, the evolution for large datasets cannot be computed at interactive speeds. It
should also be noted that the computational time of a narrow-band approach grows as the zero level set expands in the domain. Lefohn et al. [82] implemented a streaming out-of-core algorithm for solving level set evolution by packing the zero level set into a dynamic, sparse texture format and computing it on a GPU. This sparse data structure is updated using a GPU-CPU message-passing scheme to account for limited GPU texture memory and programmability that existed on GPU hardware in 2003. This technique exhibited significant (10-15x) speedups over an optimized CPU implementation and allowed near-interactive computation rates for moderately sized datasets. Unfortunately, the Lefohn et al. work has quickly become out of date with the advent of GPUs having larger texture memory, faster interfacing, and significantly more advanced directives. This topic will be addressed in section 5.1 with a new algorithm and implementation for interactive visualization and simulation of level sets using modern GPUs.

2.4.3. Velocity Functions

The level set equation (Equation 2-6) contains a velocity term $V$, which has been largely ignored up to this point. The velocity of the level set is a representation that describes the motion of the surface in space and time. This framework allows for a wide variety of deformations to be implemented by a combination of global, geometric, and image-dependent terms, depending on the application area. Sections 2.4.4 and 2.4.5 describe background information on some of these terms. Equation 2-13 gives a basic template of a velocity equation as the combination of two data-dependent terms and a surface topology term. The $D$ term is a propagating advection term scaled according to $\alpha$ in the direction of the surface normal. The term $\nabla \cdot (\nabla \phi / |\nabla \phi|)$ is the mean-curvature of the surface defined in Equation 2-12 and its influence is scaled by $\beta$. The final term $\nabla A \cdot |\nabla \phi|$ is the dot product of the gradient vector of an advection field with the surface normal, which is scaled by $\gamma$.

$$\frac{\partial \phi}{\partial t} = \nabla \phi \left[ \alpha D(I) + \beta \left( \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right) + \gamma \left( \nabla A(I) \cdot |\nabla \phi| \right) \right] \quad \text{Equation 2-13}$$
We consider velocity functions that contain terms of advection and diffusion. It is important to understand the difference between these flows in the level set context. Advective flow is a propagation of finite speed in a certain direction, while diffusive flow is defined everywhere in all directions. The numerical computation of these terms involves solving a hyperbolic PDE for advection using an upwind scheme and a parabolic PDE for diffusion using central differences. Stability is enforced by adhering to the Courant-Friedrichs-Lewy (CFL) condition, which states that numerical waves should propagate at least as fast as physical waves. Therefore, the time step used for iterating the level set must be less than the grid spacing divided by the fastest velocity term in the domain. The time step is restricted based on the velocity term as shown in Equation 2-14 where $v(i)$ is the velocity calculated at voxel $i$ and $\Delta x$, $\Delta y$, and $\Delta z$ are the grid spacing in three-dimensions.

$$\nabla T \leq \forall i \left( \frac{\max(\Delta x, \Delta y, \Delta z)}{\max(v(i))} \right)$$  \hspace{1cm} \text{Equation 2-14}

With a velocity function consisting of advective and diffusive terms, image-based scaling factors can be used to guide the terms, such as ones derived from volume attributes. Section 4.1 describes a unique set of velocity functions created for evolving surfaces to segment geologic features in seismic data.
Figure 9: Mean curvature flow shrinks high-curvature regions of an object to a single point (left to right, top to bottom).

2.4.4. Mean Curvature Flow

Level set motion by mean curvature is considered such that the interface moves in the normal direction with a velocity proportional to its curvature \( v = -b \kappa \), where \( b > 0 \) is a constant and \( \kappa \) is the mean curvature defined in Equation 2-11

\[
\kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}. \tag{Equation 2-12}
\]

For \( b > 0 \) the front moves in the direction of concavity, such that circles (in 2-D) or spheres (in 3-D) shrink to a single point and disappear \([42][43][44]\) (see Figure 9). As shown in \([121]\), oscillations on the moving front decay for this case since the total variation of the speed function for positive \( b \) has derivative \( v = -b \) and hence the total variation decays.
2.4.5. Ill-Posed Reverse Curvature Flow

The case of mean curvature flow (Section 2.4.4) where $b<0$ yields a positive speed derivative $v = b$ and causes oscillations to grow as the front moves in the direction of convexity such that circles and spheres grow. This effect causes small erroneous perturbations on the front, including those due to round-off errors, to grow into large features. It can be seen that this case is ill posed as the sign of the curvature term $b$ corresponds to the backwards heat equation $v=b\kappa$, which blows up over time if $\kappa$ is negative anywhere [121]. In the simplest case this causes a corner to form and thus the front is no longer differentiable.

Several groups have attempted to solve a modified backwards heat equation, but their results have generally required step-by-step procedures necessary to control the process, which are difficult to implement and do not work in 3-D [78] [20]. Steiner et al. [128] recognized that for a small neighborhood near the zero level set ($\phi=0$) of a simple convex curve, the outer level sets having lower curvature propagate outwards more slowly than the inner level sets with higher curvature and therefore propagate more quickly. This difference in curvatures is the reason discontinuities form on the front. In section 4.5 a technique is described to overcome these problems to allow for three-dimensional shape exaggeration by growing regions of positive curvature outwards.

2.4.6. Geometric Properties (Skeletons)

Shape extraction and surface segmentation are important topics in the fields of computer vision, visualization, and pattern recognition. The medial surface of an object provides a simple and compact representation of a 3-D shape that preserves many of the topological and size characteristics of the original. The extraction and subsequent segmentation of these medial surfaces are of interest in many application areas such as object recognition, data compression, 3-D navigation, feature extraction, and surface generation. Modern data acquisition and computation techniques have caused the generation of large, high-resolution 3-D volume data sets
such as those from MRI, CT, seismic, satellite, CAD, molecular, and scientific simulations. These new datasets contain features with interesting medial surfaces. Features such as geologic faults, car-body shells, cortical sulci ribbons, and other thin plate-like or sheet-like structures need to be analyzed in 3-D and therefore cannot be represented by the reduced dimensionality of a medial-curve. The segmentation of regions and branches from these medial surfaces is important because it allows for navigation and interpretation to follow feature characteristics using information derived from the original object.

Figure 10: 2-D example of the medial-curve for a simple rectangle shape. Left image shows the medial-curve in blue, while red points are the locus of the maximal disks given by dashed lines [9]. Right image shows the extracted medial-curve as segmented components differentiated by color.

Methods for extracting medial-curves (centerlines) and medial surfaces (center-surfaces) from 2-D and 3-D objects are generally described as skeletonization methods, which are divided into three classes [149]: topological thinning, distance-based methods, and polygon-based methods. The last class of techniques focuses on an explicit, polygonal representation of objects. Since this thesis focuses on operating with an implicit representation of an object, only the first two classes of techniques are discussed.

Distance-based methods [77][80][91][94] are continuous approaches that detect the object skeleton by looking for singularities or critical points in the distance transform of the object’s boundary. The distance transform representation is convenient for many approaches since it is the representation of an object used by level set techniques (see section 2.4). The popularity of level sets for tracking interfaces and shapes motivates the desire to continue working with a distance transform representation during medial surface extraction. It has been
shown [77][91] that this definition of an object’s skeleton is equivalent to the geometric locus of the centers of maximal disks (Figure 10) proposed by Blum and Nagel [9] and to that of the ‘prairie fire model’ proposed by Blum [8]. Continuous methods [114] based on the distance transform have advantages over topological thinning approaches [102] in that the produced skeletons are numerically accurate and complex while noisy topologies are properly handled. A brief overview of relevant works is presented that are similar in technique to what is being described in section 4.3 and, when appropriate, in their skeletonization goal of extracting a medial surface instead of a medial curve.

Kimmel et al. [77] recognized that compact boundary segments delimited by curvature extrema along the boundary on an object generate skeleton points. They build skeletons of 2-D objects from separate distance transforms of the boundary segments and then located by the zero sets of the distance transforms. Unfortunately, when a noisy boundary is present this approach has problems accurately detecting curvature extrema, and objects with holes require a special treatment. Siddiqi et al. [126] tracked marker particles while advancing the distance transform front. The skeleton was defined at the shock points where an energy conservation principal was violated. This technique is more numerically stable than direct singularity detection [12][45][94], but is difficult to implement and requires considerable effort to maintain a dense marker particle distribution for the evolving front. Rumpf and Telea [114] used scaled zero order moments to indicate singularities in the distance transform. After thresholding the singularities, another transform is solved close to the singularities and an outwards distance transform is calculated from the singularities to create an inflated skeleton. This technique requires three different distance field calculations to extract the skeleton and it does not generate orientation information or attempt to classify discrete surfaces. Hassouna and Farag [51] present a framework that uses multiple propagating wave fronts to extract skeletons, but their approach only extracts medial curves and is not easily extended to medial surfaces.
More recently, Latecki et al. [80] present a technique based on a diffused vector field that extracts a skeleton strength map based on the gradient magnitude. The resulting skeleton strength map is linked using Dijkstra’s shortest path algorithm [27]. This technique differs from the one presented in this thesis in that theirs is only defined for 2-D objects, they operate on an intermediate form of the distance transform, they do not use orientation information to guide the shortest path algorithm, and they do not attempt to segment skeleton features. Using a different technique for extracting the center-curve of an object, Reniers and Telea [108] present a technique for shape segmentation of object boundaries based on components derived from center-curve junctions. The segmentation technique presented in section 4.3 differs from Reniers and Telea in that they are focused on segmenting the original object boundary instead of the medial surface, and their segmentations were based on point-junction relationships instead of point-point relationships. In [147], Zhang et al. expand on [126] by introducing a Euclidean distance function-based method for computing and segmenting medial surfaces for matching articulated 3-D object models. Their technique is similar to the one presented in this work based on how surface points are classified prior to segmentation. An important difference is that their segmentation uses localized information on the rectangular lattice from thinning the explicit medial surface, which is difficult to implement and susceptible to noisy object boundaries [102][126].

2.4.7. Implicit Surface Visualization

Implicit surfaces are typically stored in a volumetric format similar to the one described for 3-D seismic datasets in section 2.1.1. The representation is such that voxels in 3-D correspond to discrete components of the implicit surface where the voxel value is the value of the distance transform, and the zero-valued isosurface through the volume is the level set surface. This surface is most readily visualized by using a volume cutting or clipping technique [120]. This technique is used for describing the contents of volumetric data by cutting through the volume with
orthogonal 2-D slices (e.g., Figure 3). The contents of the volume that intersect with the slice are then interpolated onto the slice. The next step is to color-map the data by assigning a color to data values on the slice. A scalar mapping is accomplished by indexing into a lookup table, which holds an array of colors that are assigned to scalar values. The effects of this mapping are adjusted by using a histogram to define the maximum and minimum scalar values of the table. Scalar values greater than or less than the range of the histogram are clamped to the maximum and minimum color values, respectively (see Figure 11). The color maps defined in a histogram represent 4 different channels: red, green, blue, and alpha. The first three red, green, and blue (RGB) channels are intrinsically mapped to a single channel and used to represent colors, while the alpha channel is used to define visual transparency or opacity. When the alpha channel is set to 1.0 pixels are rendered opaque, when the alpha channel is set to 0.0 pixels are rendered transparent, and when the alpha channel is between (0.0, 1.0) pixels are translucent allowing some light to pass through.

![Figure 11: Histogram of values for a scalar volume (blue). Gray line represents an opacity curve, rainbow colors designate color map, min and max values set the clipping range of the histogram.](image)

For the case of an implicit surface volume where voxel values represent the distance transform, the zero-valued level set surface can be visualized by adjusting the histogram to isolate the zero-values. A more explicit approach to representing a constant scalar value along a 2-D slice is to generate an isoline that contours the level set. Since the level set is a discrete set of
scalar values on the 2-D slice, the contour of interest may not exist in a single pixel but instead lie between the values of neighboring pixels. Therefore interpolation is required to generate the approximate intersection along edges of pixels. The resulting edge locations are then connected to each other, forming the isoline. In section 5.1, advanced 3-D techniques for level set surfaces are presented that expand on this section by allowing interactive visualization.

### 2.4.8. Steering

Surface editing allows users to modify a surface by hand in order to fix imperfections, errors, artifacts, or add information to the surface based on the knowledge of domain experts. Editing while a surface is evolving is called **steering**, which is an interactive form of surface editing. Steering can also be described in the context of an application that allows users to interactively control the parameters of a computational process during its execution [104][89]. This section reviews previous work in the area of implicit surface editing as background to later sections that present techniques for modifying surfaces prior to simulation (Section 5.4), during simulation (Section 5.4), and post simulation (Sections 5.3 and 5.5.3). These surface steering techniques are implemented with additional simulation and visualization controls in the IVE presented in Chapter 5.

Editing implicit surfaces is most commonly accomplished by converting to explicit surfaces such as polygonal and parametric meshes, and editing them by employing the techniques developed for editing explicit surfaces [23]. Unfortunately, this representation has the same limitations mentioned earlier for explicit surfaces, such as handling self-intersection, merging multiple meshes, and changing the genus of a surface model. In addition, converting to an explicit form requires significant effort in creating a triangulation that must be converted back to an implicit form for further processing. Therefore, we focus on techniques that allow for the editing of implicit surface models in their natural state.
When editing an implicit surface, a region of interest (ROI) must be defined in order to constrain the editing to a certain region of the surface without creating edge effects. This can be implemented by a tapering function that respects the negative inside and positive outside convention of the implicit surface and decays smoothly to zero with increasing distance from the ROI. Defining a ROI not only defines an area where editing occurs but it also implicitly defines the portion of the surface that remains constant during the editing. This limits the computation of the implicit surface to a much smaller domain and subsequently results in faster computation times. Being able to select ROIs interactively in 3-D is necessary for steering and editing complex evolving surfaces.

After defining a ROI, a number of functions can be applied in order to deform the underlying surface. Some of these deformations include growing and shrinking along the surface normal, which can be used to directly guide a surface into or away from features in the data volume. In addition, editing techniques like copying one region of the surface and pasting it onto another are useful for duplicating desired surface regions and reducing the editing time. After a surface is finished being topologically modified, it is classified into separate components. At this time it may be necessary to merge together similar components as well as break apart single components that should be classified as multiple pieces. The combination of these techniques describes a general framework (Figure 12) for implicit surface editing, as described in [90].

<table>
<thead>
<tr>
<th>Edit</th>
<th>CSG Operation</th>
<th>Volume Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY</td>
<td>Intersection, AB</td>
<td>Min(Va, Vb)</td>
</tr>
<tr>
<td>PASTE</td>
<td>Union, AB</td>
<td>Max(Va, Vb)</td>
</tr>
<tr>
<td>CUT</td>
<td>Difference, A-B</td>
<td>Min(Va, -Vb)</td>
</tr>
</tbody>
</table>

Table 1: CSG Operators of And, Or, Difference implemented as Copy, Paste, Cut respectively on two implicit level set models A and B represented by distance volumes Va and Vb with positive inside and negative outside values.
Previous work related to implicit surface editing derives from the general areas of volumetric sculpting, mesh-based surface editing, and implicit modeling. Constructive Solid Geometry (CSG) is a way to use Boolean set operations (And, Or, Difference) to combine solid 3-D primitives into complex objects consisting of multiple primitives [53][136]. The common CSG operators [90] are summarized in Table 1. The work of [144] expands on basic CSG Boolean operations and describes techniques on implicit surfaces that perform additional functions of blending and warping. In [90] a number of implicit surface editing techniques are presented for performing the basic operations of blending and smoothing as well as sharpening, openings/closings, and embossing of surface regions. Previous work has focused on the editing of static implicit surfaces that do not move. The work presented in section 5.4 expands on these techniques by allowing for the editing of dynamic implicit surfaces that are evolving and can be steered as they move.

Figure 12: Generic framework describing many possibilities for level set surface editing.
3. Structure Analysis of Seismic Data

3.1. Diffusive Smoothing

In section 2.3.1 a number of techniques in seismic data smoothing were reviewed. Unfortunately when interpreting features that extend across fault discontinuities, these approaches fail to sufficiently model the features stratigraphically, even after the effects of faulting are reversed in a volume. Therefore, it is necessary to have an anisotropic smoothing approach that allows diffusion in the orientation of seismic strata without removing faulting or blurring across strata, while still smoothing across smaller discontinuities. This section describes such an approach.

Orientation and strength information provided by the GST allows anisotropic smoothing to be conducted across discontinuities in the orientation of seismic strata. We review previous work on related anisotropic processes that reduce noise while preserving features. Perona and Malik [105] proposed a nonlinear PDE that conducts variable conductance diffusion for edge-preserving denoising. Weickert [138] proposed a similar technique called coherence enhancing diffusion, which directed diffusion only along directions defined by the structure tensor. The proposed approach modifies the coherence enhancing flow by using the Gaussian smoothed GST representation of seismic orientation and eigenvalue confidence values to drive the diffusion, following from work done by Jeong et al. [61]. This modified anisotropic diffusion is especially appealing for filtering stratigraphic features due to the nature of the governing diffusion equation. In the case of a seismic channel, cut by a discontinuous event such as faulting, the goal of diffusion filtering is to provide a mass transfer of depositional sediment from a region of high concentration (where the channel is continuously imaged in seismic data) to an area of low concentration (where the channel was cut by a fault and is now poorly imaged in seismic data).
This description is analogous to the way in which the image is smoothed. The approach has the dual advantage of enhancing channel edges and removing noise while smoothing discontinuities in the volume caused by faulting.

The diffusion equation is defined as follows:

$$\frac{\partial}{\partial t} I(x,y,z,t) = \nabla \cdot \left( d(x,y,z,t) \nabla I(x,y,z,t) \right)$$  \hspace{1cm} \text{Equation 3-1}

where $I(x,y,z,t)$ is the input image at iteration $t$ and $d(x,y,z,t)$ is the monotonically decreasing diffusion function of the image gradient magnitude (Equation 3-3):

$$d(x,y,z,t) = e^{-\frac{\|\nabla I\|^2}{2k^2}},$$  \hspace{1cm} \text{Equation 3-2}

where the free parameter $k$ is the conductance parameter. Diffusion is applied in the direction of the 2nd and 3rd eigenvectors of the GST, which correspond to the orientation of seismic strata as described in section 2.3.2. Since the 2nd and 3rd eigenvectors define the local stratigraphic horizon, this filtering technique can be called stratigraphy preserving, since filtering does not blur into depositional layers above or below it and therefore maintains consistent stratigraphic layering.

Figure 13 shows a 2-D stratal slice of an original seismic image, a statistical median filter result, and the results of the described filtering method (all of which have been applied in three-dimensions). This slice shows a large channel moving through the volume as can be seen by the low amplitude (blue) feature, as well as a number of smaller channel features. When comparing the original, the median filtered, and the stratigraphy-preserving anisotropic diffusion (left, middle, and right, respectively), it can be seen that the stratigraphy preserving anisotropic diffusion filters out much of the noise contained in the channel features, while still preserving the precise location of channel boundaries as they appear in the original data. Channel boundaries are preserved by the diffusion function in Equation 3-2 such that when the gradient magnitude is large, as occurs on the edges of channels, no smoothing is applied.
Figure 13: Slice of an original 3-D seismic image (left), statistically smoothed using a median filter (middle), and the stratigraphy-preserving anisotropically diffused image (right). Zooming in on the large channel shows the median filter shrunk the width of the channel (middle, bottom) while the stratigraphy-preserving anisotropic diffusion retained the original size width of the channel (right, bottom). In addition, other channel features visible in the original data as shown by the yellow and brown arrows begin to disappear when smoothed using median filtering (middle), yet they are preserved and enhanced in the anisotropically diffused data (right).

3.2. Critical Points

In the previous section (3.1) a technique was described using structure analysis for removing noise in seismic data by conducting anisotropic smoothing along stratigraphic layers. The result is a new seismic volume with attenuated noise and enhanced features. The next step is to use structure analysis for extracting information that helps identify data features. First, a more robust representation of the orientation field given by the structure tensor is computed using Gaussian convolution, which averages the tensor orientations (see section 2.3). Next, the eigenanalysis of the smoothed structure tensor can be computed in order to provide the local orientations as well
as indications of singularities in the data volume. The GST has three real eigenvalues and corresponding eigenvectors. The eigenvectors define a local coordinate axis while eigenvalues describe the local coherence, which represents the strength of the gradient along each respective axis. Potential critical points are located in the data volume by using the three-dimensional gradient magnitude given by Equation 3-3.

\[
|\nabla I| = \sqrt{I_x^2 + I_y^2 + I_z^2}
\]

Equation 3-3

The gradient magnitude is a simple and powerful technique for detecting singularities. When isolating medial-surface components in a distance transform volume (section 4.3), singularities are defined by areas of low gradient magnitude. The opposite (i.e., high gradient magnitude) is used when identifying channel edges from a seismic volume (3.3). Isolated singularities can be classified as 1-saddles, 2-saddles, and maximums as depicted in Figure 14. These three types of singularities (or critical points) are classified by their eigenstructure as:

1-Saddle: \( \lambda_1 >> \lambda_2 \approx \lambda_3 \)

2-Saddle: \( \lambda_1 \approx \lambda_2 >> \lambda_3 \)

Maximum: \( \lambda_1 = \lambda_2 = \lambda_3 \),

where \( \lambda_1, \lambda_2, \lambda_3 \) are the three eigenvalues of the structure tensor in descending order. In the context of classifying medial-surface components, the dominant eigenvector of a 1-saddle represents the orientation of the gradient orthogonal to the surface, while the smaller two eigenvectors form an orthogonal plane parallel to the surface. For a 2-saddle, the two most dominant eigenvectors represent the gradient orientation of the surface and the smallest eigenvector represents the orientation parallel to the surface. A maximum critical point is characterized by an incoherent or chaotic eigenstructure with no dominant orientation. These three structures can properly identify all critical points from a 3-D object, and will later be used to identify and classify medial surfaces of level sets, as is discussed in section 4.3.
Figure 14: (a): 1-Saddle, (b): 2-Saddle, and (c) Maximum critical points of a surface in 3D. Bottom image gives examples of each critical point type in a seismic fault dataset.

### 3.3. Confidence and Curvature

A structure analysis technique is now described to specifically enhance geologic channels in seismic data. Previously, more general approaches to enhancing and locating features using the first-order structure tensor were described [7]. Here a mathematical model given by the second order tensor (Equation 2-5), called the Hessian matrix, is used to enhance translation invariant second order structures in a seismic dataset. This technique operates under the assumption that seismic data has already been smoothed to enhance continuous stratigraphy more than small discontinuities (section 3.1).
Frangi et al. [41] exploited the eigenvalues of the Hessian in order to develop an MRI vessel enhancement filter. Their main contribution is a vesselness function that integrates all three eigenvalues computed at a range of scales in order to detect various vessel sizes. Sato et al. [118] expanded on their work and used the Hessian to detect sheets and blobs in 3-D images. Seismic channels represent a domain-specific image feature that cannot be appropriately modeled using either of these techniques based solely on Hessian analysis because of a channel’s unique layered structure (Figure 8). To address this difficult problem, we develop a new confidence and curvature-based attribute that is able to enhance channel features and provide input to the segmentation PDE (section 4.1).

Bakker et al. [7] detected channels in 3-D seismic datasets by using the first order structure tensor (GST) to identify the location of features while honoring seismic orientation. In particular, they used an orientated GST and enhanced features while removing noise by filtered eigenanalysis. Through their orientated representation, they were able to create a curvature-corrected structure tensor that accounted for line-like and plane-like curvilinear structures. They attain a confidence measure from the eigenvalues of the transformed GST, where larger eigenvalues correlate to increased confidence in the orientation estimation. Their primary contribution is the development of an approach to extract curvature information using a parabolic transformation of the GST, which yields a curvature-corrected confidence measure that is maximized for the transformation most closely resembling the local structure.

The method presented in this work is similar to that of Bakker et al. in how confidence and curvature information is obtained from image structure analysis. However, there is a significant difference in the approach presented here since it employs the second order tensor to directly extract confidence and curvature information instead of relying on an intermediate transformation as done by the Bakker et al. method. The second order tensor has the advantage of directly providing this information without needing to use a parabolic transformation. Concerns are often raised about the error in second order calculations, which can result in unstable tensor
fields. However, this problem is overcome by applying tensor smoothing across the volume using a Gaussian kernel, which stabilizes the tensor components without destroying the Hessian representation. The confidence and curvature information is later used to drive a segmentation process for completely extracting channel features (section 4.1), which is something that was not considered by Bakker et al. [7] or in previous work.

A measure of confidence and curvature in seismic data corresponds to regions of high depositional curvature that present a strong and confident amplitude response. As described in section 2.2.4, this description maps well to the imaging of stratigraphic features such as channels. The goal is to define a \textit{channelness} measure that captures the specific structure associated with channels. The first eigenvector $v_1$, and its corresponding eigenvalue $\lambda_1$, are a primary focus. Due to the layered structure of channels (see Figure 8), they are approximated as planar features with high curvature along the gradient direction (Figure 14a), which corresponds to the first eigenvector. Therefore, by comparing the first eigenvector to the second, a \textit{channelness} measure is defined in Equation 3-4 as the difference of the first eigenvalue $\lambda_1$ with the second $\lambda_2$ scaled by the mean average of all $\lambda_i$:

$$C(I(x,y,z)) = \sum_{i=1}^{n} \frac{\lambda_i - \lambda_2}{\lambda_i + \lambda_2}. \quad \text{Equation 3-4}$$

Since channels generally have a relatively constant cross section, the second order tensor is smoothed with a single Gaussian sigma value. Choosing a sigma value that approximates the distance across a channel results in optimal enhancement. Figure 15 shows stratal slices displaying the \textit{channelness} attribute on three different data sets. This \textit{channelness} measure was first presented in [65] and expanded upon in [70]. Section 4.1 describes a unique form of the level set equation driven by this \textit{channelness} measure specifically for segmenting channel features using second order tensors derived from seismic images.
Figure 15: Channelness measure calculated in 3-D on the stratal slice shown on the left, with the resulting attribute on the right. Bright values correspond to a high likelihood of a channel. Channel edges can be found by computing the gradient of the attribute.

### 3.4. Direct Seismic Fault Enhancement

A technique is now described to enhance fault features (section 2.2.3) directly from the seismic volume using the first-order structure tensor. There is a long established line of research in seismic interpretation that has lead to the characterization of geologic faults based on their structural character \[5\][61][23]: faults are discontinuities in the strata that extend vertically. This characterization is the motivation behind a function that returns positive propagation values for features and negative propagation values for non-features.

Typically, faults are enhanced from raw seismic datasets using a 3-step approach \[68\]: vertical discontinuities are detected, vertical discontinuities are enhanced laterally in 2-D, and then they are enhanced again laterally and vertically in 3-D. While this is an over-simplification
of the fault enhancement technique, the key observation is that faults are never enhanced directly from a seismic volume. Instead, a number of cascaded techniques are used to create a final volume that measures fault likelihood. An effective implementation of this technique provided by TerraSpark Geosciences [68] generates a measure of Fault Enhancement, which is a probability that represents the likelihood for a fault to exist at a given voxel in the volume. The problem with using a number of cascaded attribute volumes to enhance fault structure is that incorrect information can be added anywhere along the pipeline and hence it is difficult to identify the source of this bad information. Although these cascaded techniques are computationally efficient and produce reliable representations of fault features, it is still beneficial to generalize the approach to a single function that can be computed directly from the seismic data:

\[
\begin{align*}
\text{var}(x, y, z) &= \text{plane}(\vec{v}_2 \times \vec{v}_3) \\
f(x, y, z) &= \sum_{i=-n}^{n} \text{var}(x + i\vec{v}_1^x, y + i\vec{v}_1^y, z + i\vec{v}_1^z).
\end{align*}
\]

Equation 3-5 follows directly from work done by Jeong et al. in [61], as well as other proprietary algorithms in industry. Therefore, it is desired to directly enhance faults from the seismic data using a single function. This can be accomplished by looking for discontinuous features in the seismic strata by calculating the variance across the strata. In particular, discontinuities can be located along the seismic strata defined by the two smaller eigenvectors of the structure tensor (see section 3.1). Given this representation, the first step is to compute the variance within a user-defined planar window along the strata of the voxel under consideration. Next, moving along the positive and negative direction of the dominant eigenvector and using the same planar window, additional variances are calculated and summed together. The summation of these variances completes the fault attribute computation. Other fault imaging techniques, such as coherence, semblance, edge-stacking, or continuity, follow a similar approach and achieve comparable results in recognizing these discontinuous regions. What makes this approach unique
is that the local strata is used to guide the vertical summation of variances, which is different from traditional approaches [23] that make an assumption of perfectly horizontal strata layering.

Function $f(x,y,z)$ in Equation 3-5 results in a scalar field such that higher values correspond to a greater likelihood of a fault and lower values correspond to a lower likelihood of a fault. It will be shown in section 4.2 that this analysis of discontinuities in seismic data can be used to directly guide level sets for fault extraction. In section 5.1.9.2 this technique is used for the enhancement of fault features calculated on the fly directly in seismic data during level set surface evolution.

![Figure 16: Computation of vertically summed variance along the strata for fault imaging.](image)

### 3.5. Facies for Seismic Segmentation

Previously, techniques were presented for enhancing features in seismic datasets using tensor analysis. There also exists a great potential in using the structure of seismic data for modeling a number of other geologic features so that they can be segmented with implicit surfaces. This remaining area can be generalized as the analysis of seismic facies. A review of work in this area was completed in 2005 by the European research and training network NetAGES (Network for Automated Geometry Extraction from Seismic) and the collection appears in [58]. This section describes the basic formulations for enhancing other geologic features using the structural analysis of seismic data.
Figure 17: Volume segmentation of a 1-D signal using the watershed algorithm. From left to right, watershedding is implemented by slowly raising a water level (blue) such that basins are submerged. When the water originating from two different basins comes together, a watershed is created to divide the regions (red dashed line). At the completion of the process, regions are automatically classified by the differentiating watersheds (colored blocks).

Seismic facies can be described as sedimentary textures in seismic data that are different from adjacent textures in their data characteristics [113]. The characteristics generally considered are amplitude, frequency, polarity, continuity, reflector configuration, reflector abundance, geometry of textures, and the relationship to neighbors. The structure analysis of seismic data described in this work creates attributes that can be used for analyzing and classifying these seismic facies with the help of volume segmentation techniques. Volume segmentation [127] is a common technique for dividing an attribute volume into different regions, and it can be implemented using the watershed algorithm (Figure 17).

The watershed algorithm separates a volume into regions based on the grayscale value of an attribute. Applying the algorithm results in the creation of regions called basins and delineations called watersheds that delineate discrete regions. No improvements to the basic watershed algorithm [58] were made, however our contribution is to observe that the watersheds can be used for simplifying structural attributes of seismic data (see Figure 18). For instance, using the magnitude of the gradient calculated by the GST as an input to the watershed algorithm allows the segmentation of a seismic volume along stratigraphic layers (Figure 18, middle). If trying to segment regions of the volume delineated by faults, the measure described in section 3.4 can be used as a watershed feature that creates fault “blocks” (Figure 18, right). Segmenting the structural analysis of a seismic volume in this way allows for the creation of terms that can be used in a multi-attribute level set segmentation. Section 4.6 describes the application of this
volumetric representation to provide inputs to a multi-attribute level set process that may be useful for the modeling of geologic reservoirs.

Figure 18: Watershed segmentation of a seismic volume (left) into horizon intervals (middle) and fault “blocks” (right).
4. Level Set Techniques for Geologic Surfaces

In this section, the structure analysis work from chapter 1 is presented as a way to drive a number of novel level set representations. The level set techniques were developed in order to solve specific problems in seismic interpretation, but they have in turn become general techniques useful to a wide variety of implicit modeling problems.

4.1. Confidence and Curvature Guided Level Sets

This section presents a new method for segmenting channel features from 3-D seismic volumes. Parts of this section derive from work presented in [65] and [70]. Previously in section 3.3, the strength and direction of second-order eigenvectors were used to enhance channel features by generating a confidence and curvature attribute. Now, that tensor-derived attribute is used to form the terms of a PDE that is iteratively updated using the level set method. Results from this technique are shown on two seismic volumes in order to demonstrate the effectiveness of the approach.
Figure 19: Computation of the inside of a channel by identifying high curvature on lateral slices (left) and location of channel edges based on the gradient on the boundary of a channel (right).

The confidence and curvature analysis of the Hessian allows for the volumetric enhancement of features (section 3.3), but it does not complete the segmentation required to fully represent a channel. Recall that 3-D image segmentation can be accomplished explicitly in the form of a parameterized surface [117] or implicitly in the form of a level set [97]. As described in section 2.4, the level set is the preferred technique because of its ability to handle complex geometries and topological changes. The level set method requires additional information about regions to be segmented in order to drive the propagation of the implicit surface. This is commonly done in the form of a scalar speed function that defines propagation speeds in the surface normal direction. Feddern et al. [39] recently described a structure tensor valued
extension of curvature flow for level sets. Their work generalized the use of the structure tensor for mean curvature flow by utilizing image tensor components in the curvature calculation. The work presented here expands on their generalization by allowing a level set surface to evolve toward specific features using a propagation speed given by a tensor-derived channelness term, an advection motion also based on the channelness term, and mean-curvature motion to encourage a smooth final segmentation.

In order to guide the level set evolution towards channel features, the velocity equation consists of two data-dependent terms and the mean-curvature term. The level set evolution is therefore defined as the combination of three terms as shown in Equation 4-1:

$$\frac{\partial \phi}{\partial t} = \nabla \phi \left( \alpha D(I) + \beta (\nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}) \right) + \gamma (\nabla A(I) \cdot |\nabla \phi|) \quad \text{Equation 4-1}$$

The $D$ term is a propagating speed term defined by the channelness (Equation 3-4) and scaled according to $\alpha$ in the direction of the surface normal. The term $\nabla \cdot (\nabla \phi |\nabla \phi|)$ is the mean-curvature of the surface and its influence is scaled by $\beta$. The final term $\nabla A \cdot |\nabla \phi|$ is the dot product of the gradient vector of the advecting force, defined as inverse channelness, with the surface normal. The advecting inverse channelness gradient is scaled by $\gamma$. The contribution of each of these terms is generalized in Figure 20 for a simple 2-D segmentation example of evolving a shape towards a bright feature as in ITK-SNAP [itksnap.org].

![Figure 20: Visual representation of the contribution of level set terms in equation Equation 4-1 for evolving a surface (or contour) towards a bright intensity feature in 2-D.](image)

The combination of two image-fitting functions with a mean-curvature term is necessary to achieve realistic channel segmentation. The image-fitting functions cause the surface to flow
into channel features while the mean-curvature term maintains a smooth representation and prevents the surface from leaking into non-feature regions. The propagating channelness term is derived from the second order structure tensor and drives the segmentation into regions with a high likelihood of containing a channel feature. This representation is appealing as the physical process being calculated in this term can be interpreted as an external convection field. Although far from realistically simulating the ancient fluid flow that created the channel, the channelness guided propagation follows convective laws used in the erosion and deposition of a flowing medium and therefore has physical meaning. As channelness highlights the interior of a channel, the gradient of its inverse highlights feature boundaries and edges. Using this gradient as an advecting force represents the way in which the evolving surface moves toward channel edges when parallel to them, but does not cross over the edge. When driven by the channelness propagation, this advecting force acts like the bank of an ancient channel where the flowing medium is forced to stop and then move parallel along the edge. The mean-curvature of the surface is useful for alleviating the effects of noise in the image by preventing the surface from leaking into non-channel regions and maintaining a smooth representation. The combined contribution of these terms can be adjusted using the $\alpha$, $\beta$, and $\gamma$ constants according to the nature of the feature being segmented. In general, an equal contribution value of 1/3 for each term is sufficient to accurately segment the channel. In the case of a greatly meandering channel, the mean-curvature term ($\gamma$) should be de-emphasized in order to allow a more sinuous segmentation.

Figure 21: Slice of channelness attribute of 3-D seismic volume overlaid by the red outline of the level set segmentation. Left to right, increasing iterations of 10, 50, 100 respectively.
The results of segmentation using confidence and curvature-guided level sets are shown for channels from two different 3-D seismic volumes. In practice, geoscientists prefer to manually define the centerline of a channel they hope to segment since it is a relatively quick step compared to manually interpreting the entire 3-D channel surface, which requires significantly more time. For this reason, the initial seed used in each of the segmentations was a 1-pixel wide tube manually drawn to approximate the center of the channel from end to end. Level set seeding is discussed in further detail in section 5.2.

Figure 22: Slice of channelness attribute of meandering channel from 3-D seismic volume, overlaid by the red outline of the level set segmentation. Left to right, increasing iterations of 10, 50, 100 respectively.

The channel in Figure 21 is cut by discontinuities (faults), which can be seen on the time slice view as bright isotropic regions. The image was first anisotropically diffused (section 3.1) along the seismic strata, which improved imaging near the discontinuities to create a more continuous image of the channel. Next, the image was segmented using the level set equation described in Equation 4-1. That resulted in the 3-D representation of the channel shown in Figure 23. It should be noted that this surface is the result of applying the method with a Gaussian sigma of 5.0 for smoothing the structure tensors and equal scaling values used for $\alpha$, $\beta$, and $\gamma$ of the level set evolution equation.
Figure 23: Three-dimensional representation of a segmented channel displayed in different orientations on a y,z-slice (top left), y- and z-slice (top right), x- and z-slice (bottom left), x- and z-slice rotated (bottom right).

Figure 24: Original seismic image on a z-slice different from Figure 11 (top left) and three-dimensional representation of the segmented channel displayed on the z-slice (top right), x- and on z-slice (bottom left), x- and z-slice rotated (bottom right).

The channel shown in Figure 22 is a narrow meandering channel. Enhancing this channel requires a smaller Gaussian sigma of 2.0 and a $\beta$ value approximately half the size of $\alpha$ and $\gamma$. As mentioned above, the $\beta$ value for the mean-curvature should be adjusted with respect to the $\alpha$ and $\gamma$ values depending on the channel that is being segmented. Since this channel is more sinuous, decreasing the influence of the mean-curvature term allows it to be treated as such. Figure 24
shows a different slice of the original 3-D volume, and the 3-D segmentation of the meandering channel at different rotations.

In general, for this application, it is not desired to have a single parameter-set used for all channels, since over geologic time channels often deposit on top of one another. When this happens it becomes necessary to differentiate between two intersecting features by manually choosing these parameters. For this reason, the technique was developed such that a limited amount of user-control is necessary in order to allow semi-automated segmentation of channels.

### 4.2. Planar Level Sets

An approach is now described for representing planar level sets in 3-D for the purposes of fault segmentation. This is challenging for implicit surface modeling since a planar fault surface is a 2-D manifold in three-dimensions, which is both difficult to represent and compute. The reason for these problems is that derivatives are not defined everywhere on the fault manifold, for instance at the edges of the fault manifold, and that implicit surfaces require an inside and outside of a surface to be defined, but a manifold has no inside points. Therefore, the approach was taken to represent a segmented fault as the bounding surface of the fault’s 2-D manifold (see Figure 25). This representation allows curvature to be defined at all points of the segmentation so that the actual fault surface can be segmented by a medial-surface extraction (section 4.3). An additional advantage to representing the segmented fault as a bounding surface is that it approximates a region called the fault damage zone, which is used by geoscientists for conducting reservoir modeling.
In this figure two views of the bounding surface of a fault’s 2-D manifold are shown. Colored surfaces represent the actual 2-D fault manifold and silver surfaces are the bounding surface of the fault.

The starting point for segmenting faults is the initial seeds, which are assumed to be either manually picked or automatically extracted. Level set seeding is covered in more detail in section 5.2. The initial seeds are first represented as an implicit surface, which requires a velocity function to drive growth for the accurate segmentation of faults. A natural representation for this function can be derived from the approaches described in section 3.4. Given the success gained from using a fault likelihood measure for highlighting faults [68], this measurement is used as a basis for the level set velocity function. The fault likelihood is a scalar byte value $f$ from (0-255) and it can be thresholded for the level set velocity in a number of different ways. The goal of thresholding on the fault likelihood is to encourage growth in regions of high fault likelihood and shrinking in regions of low fault likelihood, as done for tumors in [82]. The $T$ term in the fault likelihood function specifies a threshold value around which faults are segmented. For the case of the sawtooth form (Equation 4-2), all voxels in the volume greater than $T$ will grow and all voxels less than $T$ will contract the level set. The result of the corresponding speed function is shown in Figure 26.

$$F(i) = i - T \quad \text{Equation 4-2}$$
The threshold-based speed function is combined into the level set equation given as:

\[
\frac{\partial \phi}{\partial t} = \nabla \phi \left[ \alpha F(I) + \beta \left( \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right) \right]
\]  

Equation 4-3

Where \( F(I) \) is the fault likelihood propagation function on volume \( I \) scaled by \( \alpha \). The term \( \nabla \cdot (\nabla \phi / |\nabla \phi|) \) is the mean curvature of the level set, scaled by \( \beta \). As in other level set velocity functions (section 2.4.3), the coefficients \( \alpha \) and \( \beta \) designate the amount of influence the terms of the equation have on the overall growth process. This velocity equation becomes more advanced with the addition of a feature exaggeration term as will be covered in section 4.5, and using generalized advection constraints discussed in section 4.4.

When level set growth is determined by parameters of fault likelihood and mean curvature, a challenge exists to determine the proper weighting of these terms in the velocity calculation. The tradeoff is to prevent leaking growth of the fault into undesirable regions while still allowing controlled growth into faulted regions, and is controlled by the coefficients \( \alpha \) and \( \beta \). Determining the optimal values of these coefficients required significant testing on a number of different data sets in order to properly model the behavior of fault growth. Computing multiple iterations of the level set evolution with a range of coefficient values allowed for a determination.
of which coefficients produced the best growth. Figure 27 shows one time-slice view from iteration 0 and one slice at iteration 100 of a fault-likelihood based simulation where curvature had an effect of $\beta=0.05$ and fault-likelihood an effect of $\alpha=1.0$. Figure 28 shows one time-slice view from a dataset at iteration 0 and two slices at iteration 100. In one case (left) high propagation was conducted ($\alpha=1.0$, $\beta=0.0$) and in the other case (right) high propagation was balanced with curvature ($\alpha=1.0$, $\beta=1.0$). It can be seen that the curvature term has a regulating effect on the flow and maintains a more smooth evolution. It should be noted that it is convenient to show 2-D time-slice images on paper to describe the growth of fault evolution, although it is important to remember that this simulation is happening in 3-D. After completing tests on a variety of datasets, the optimal starting choice of these coefficients was determined to be $\alpha=0.125$ and $\beta=1.0$. Any changes made to the velocity equation will result in a change of these coefficients and different datasets will likely require reconsideration of these values.

Figure 27: Example of high-propagation evolution for (left) initial and (right) final time steps. Background grayscale image is the fault-likelihood data overlaid in red by the level set fault extraction. Black arrows points to initial seeds that shrunk and yellow arrow points to a new fault region that the technique discovered.
Figure 28: Comparing propagation only flow (left) to propagation flow with curvature flow (right) for the initial seeds (top). Blue represents the level set surface and red is the boundary of the surface. Bright features in the background image are faults and dark features are non-faults.

In analyzing the results of this process, the advantages of using the level set representation for segmenting fault features should be noted. In Figure 27, black arrows represent initial features in the seed level set that did not grow into faults, or in other words, they shrunk. The yellow arrow points to a feature that was not found in the initial seed image, but after sufficient iterations (200 in this case), the level set evolution was able to expand into this fault region. In later sections, Figure 29, Figure 35, Figure 54, and Figure 71 show these results in three-dimensions in order to describe more intuitively what this technique is accomplishing and the complexity of fault structures (i.e., intersecting and X-patterns) it is able to represent. Much of the work done in this section was presented at the 2008 AAPG Annual Meeting on the topic of
using level sets for fault extraction [68] and appeared in the Springer Journal of Visual Geosciences [69].

![Figure 29: Fault surface evolving in 3-D from initial seed lineaments (left to right), for 50 iterations, 100 iterations, then the segmented medial-surfaces of the fault surface (right), as will be described in section 4.3.](image)

### 4.3. **Medial-Surfaces**

The previous section 4.2 described a technique for representing planar level set surfaces for the purposes of fault segmentation. Unfortunately, the representation for planar level set evolution does not represent the solution as a 2-D manifold in three-dimensions, instead it is the bounded surface of the fault manifold (see Figure 25). Although this bounded surface is useful for modeling a fault damage zone, it often is necessary to represent a fault as a planar manifold. Therefore, a technique was devised for extracting the *medial surface* of a planar level set surface, which can then be represented as the fault manifold. It is desired to not only extract but to also segment or classify the resulting manifold into discrete components. A novel technique is presented for accomplishing this task as well. Background information on medial-surface extraction was presented in 2.4.6. The contents of this section were taken largely from a paper presented at the 2008 IASTED Visualization, Imaging, and Image Processing [66].

The technique described in this section calculates the structure tensor across the distance transform of a level set volume, such as one output from a planar fault evolution (see section 4.2). Structure tensor orientations are first Gaussian smoothed to generate a stable tensor representation across the object. Next, critical points are computed in the distance transform using a gradient magnitude calculation and classified by their eigenstructure (section 3.2). Medial surface thinness is attained by non-minimal suppression along the direction of low gradient magnitude regions.
Next, to guarantee a continuous surface, connectedness of the medial surface is computed by a modified shortest path algorithm that links remaining surface regions along the orientation defined by the structure tensor. Finally, orientation information that was extracted from the structure tensor is used as a constraint to a hierarchical segmentation algorithm, which partitions the medial surface into meaningful components such as co-planar surfaces and junction regions.

This approach brings together many techniques presented earlier in this work such as structure tensor analysis in section 2.3, theory for level set distance transforms from section 2.4.6, and critical point detection in seismic volumes from 3.2. In what follows, it is assumed the structure tensor has been previously calculated to locate singularities and orientations in the distance transform volume. This information is used to create a thin and connected medial surface, which is further segmented into meaningful components.

4.3.1. Thinness and Connectedness

After locating critical points in the distance transform to represent an initial medial surface, thinness is attained by performing non-minimal suppression at each critical point. Suppression is performed along a critical point’s gradient orientation, since it is orthogonal to the orientation of the medial surface. Removing points along this orientation guarantee that thinning removes only points that shouldn’t be included in the final surface.

Non-minimal suppression checks two neighbors for a 1-saddle and eight neighbors for a 2-saddle, and removes points that have a gradient magnitude greater-than or equal-to the critical point under consideration. Non-minimal suppression is not performed for maximum points, since there is no gradient direction at those points (Figure 14c). This technique is motivated by the many surface thinning approaches used for medial surface extraction [81][116][102], but its use here only allows small and localized thinning to occur (see Figure 30). Since local structure information is known at each identified critical point, thanks to the structure tensor, thinning is directed only along directions that do not enhance the local medial surface. Enacting thinning at
localized regions prevents the common problems associated with multiple subiteration algorithms [102] such as slow running time and incorrectly biasing the location of the medial surface.

![Image](image1.png)

Figure 30: Removal of non-medial surface critical points from the distance transform. Non-minimal suppression thinning is conducted using the gradient structure tensor to preserve the medial surface. (Top): Before non-minimal suppression and (Bottom): after non-minimal suppression, which results in a thin representation of the surface.

Connectedness of the medial surface ensures that gaps created incorrectly during the thinning process are filled. This is accomplished by a modified shortest path algorithm that links surface regions starting in the direction along the medial surface orientation defined by the structure tensor. Using the same eigenvectors calculated previously, neighboring medial surface critical points are defined as eight connected neighbors for a 1-saddle and two connected neighbors for a 2-saddle. Maximum critical points are only considered for search termination
points since they have no orientation information to guide a search. Multiple shortest paths are computed using 3-D chain codes [75] along the medial surface direction defined by the structure tensor, and a search is made for the nearest two or eight critical points for 1-saddle and 2-saddles, respectively. The search is not conducted in directions along the existing medial surface already connected to the point under consideration since that direction is already connected. The search will either find other 1-saddle and 2-saddle points or find maximum critical points. A path length is defined based on gradient magnitude weighted Euclidean distance along a path \( R = \{ r_1, r_2, \ldots, r_n \} \), such that

\[
|R| = \sum_{i=1}^{n} |L(r_i)| \cdot g_i,
\]

Equation 4-4

where \( L(r_i) \) is the distance measure and \( g_i \) is the gradient magnitude at the \( i \)th point along the path, as defined in Equation 3-3. The path \( R \) corresponds to a path on the medial surface and all intermediate points on the path between critical points are added to the set of medial surface points. The search proceeds according to a dilated medial surface direction defined by the structure tensor, which prevents loops and indirect paths from being generated toward a critical point. Dilating the search direction creates multiple paths with a greater likelihood of finding nearest critical points; when multiple paths find the same critical point the shortest weighted distance is taken.
Figure 31: Connectedness search along the medial surface orientation as defined by the 3D structure tensor. Shortest path is a function of Euclidean distance and gradient magnitude. Top is before and bottom is after searching on a synthetic fault dataset.

When implemented, the connectedness step introduces a free-parameter $D$ that is the maximum path length allowed to search for a critical point along the medial surface direction. $D$ creates a heuristic to stop path-searches along directions where another critical point has a low probability of being found. However, if the value $D$ set too low it can prevent certain gaps in the medial surface from connecting.

This connection scheme is more robust than the naïve shortest-path implementation since the only paths traversed are those defined by the orientation of the smoothed structure tensor for the calculation of a critical point. This prevents spurious branches and incorrect paths from being taken along false local maximums resulting from noise in the distance transform when connecting multiple critical points. Connected neighbors are only traversed if they are not already critical points, which adds an additional shortcut that speeds up the scheme. The computational speed is
significantly improved since all 26 connected neighbors never need to be searched when looking for nearby critical points. Figure 31 attempts to visually simplify the idea of the connection scheme using a synthetic seismic fault dataset where medial surfaces are in blue, red points are critical points under consideration, dotted green arrows are failed search paths, and solid green arrows are successful search connection paths. New connections that were made between discrete surface regions are represented as green surfaces in the bottom image.

4.3.2. Clustering and Segmentation

While points are traversed in the shortest-path algorithm from the previous section, the distances and dot products of the eigenvectors between critical points are being calculated into a dissimilarity measure $M$ defined as:

\[
\begin{align*}
    d &= \sqrt{(x^a - x^b)^2 + (y^a - y^b)^2 + (z^a - z^b)^2} \\
    p &= e_{1a} \cdot e_{1b} \\
    M &= d \frac{1}{p}
\end{align*}
\]

Equation 4-5

where $x^a$ is the x-coordinate of point $a$, $e_{1a}$ is the first eigenvector of point $a$, and $M$ is the product of the distance $d$ between points and the inverse dot-product $p$. When $d$ is small and $p$ is close to 1, points $a$ and $b$ are similar since they are close to each other and have a nearly identical eigenstructure. This dissimilarity measure is calculated first for the points connected in the shortest-path algorithm followed by 1- and 2-saddle critical points within a certain nearest neighbor distance of each critical point accepted to the medial surface. These points are then stored in a min-heap ordered by dissimilarity measure. Next, a merge process pops the least dissimilar (most similar) pairs of points off the min-heap and either creates a new component out of the two points if neither has been previously merged, or merges both points into the larger that is already merged into a component. In the case where both points are already merged, the smaller component is merged into the larger component as long as the average dissimilarity
between points is within a first threshold, called $\alpha$. This process proceeds in a hierarchical fashion until the dissimilarity between a pair of points or a newly merged point with a component becomes greater than $\alpha$. Adjusting the value of $\alpha$ allows the medial surface to be segmented at different levels of detail.

Figure 32: Segmentation of the medial surface on a synthetic fault dataset. Colored surfaces represent merged components, circles represent the critical point under consideration, and arrows describe eigenstructure.

Since there is a poorly defined eigenstructure at maximum points, they are segmented apart from 1-saddles and 2-saddles using a modified dissimilarity measure where $p$ is 1.0. Because maximum points describe the junction of two different structures in the object, it makes sense to keep them separated from 1-saddle and 2-saddle points so that coherent joint structures can be segmented from the rest of the medial surface components.

Conducting the segmentation during the medial surface extraction takes advantage of smoothed tensors that are calculated implicitly on the distance transform volume using neighboring non-skeleton points. This approach creates a much more robust orientation estimate than would be possible by segmenting the medial surface explicitly in post processing where the representation is less continuous since it would have to be discretized. Moreover, since this approach classifies maximum points during the medial surface extraction, junction points are readily available and can be treated separately from the rest of the medial surface. Separating these junctions from the rest of the medial surface helps segment meaningful components on
either side of a junction while providing an easy segmentation of the junction components themselves.

Figure 33: Medial-surface extraction and segmentation showing (a): original ‘hole cube’ shape, (b): distance transform, (c): medial surface on 2D slices, (d) maximum components, (e): 1-saddle components, (f): 2-saddle components.

**4.3.3. Extraction and Segmentation Results**

To test the effectiveness of this technique it was applied to two synthetic 3-D shapes containing sheet-like skeletons and two real-world 3-D seismic datasets containing a large number of faults. The synthetic shapes were chosen to show that the technique can correctly extract and segment the medial surface of an object into meaningful components by 1-saddle, 2-saddle, and maximum points. Results from the seismic fault data was selected to test robustness to noisy and complex datasets. Results display segmented medial surface components as point sets with no smoothing or surface-fitting applied, since this allows the most accurate representation of the resulting data without adding visual modifications. It should be noted that generating smoothed surfaces of the point sets will appear to be more continuous than visualizing the raw points. When representing the distance transform volume as 2-D slices bright regions correspond to greater distances, and when representing the medial surface as 2-D slices red pixels represent 1-saddles, green pixels represent 2-saddles, and blue pixels represent maximums.
In Figure 33, a 3-D hole cube is shown as was used in [102] and in Figure 34 a 3-D flat snake is shown, both are followed by their resulting distance transform, medial surfaces, and segmented medial surface components. It can be seen that the results accurately represent the medial surface of these simple objects and properly segment their medial surfaces into components.

Figure 34: Medial-surface extraction and segmentation showing (a): original ‘flat snake’ shape, (b): rotated original, (c): distance transform volume, (d): medial surface on 2-D slices, (e): rotated medial surface with different 2-D slices, (f): maximum components (very small), (g): 1-saddle components, (h): 2-saddle components, (i): rotated 1-saddle, 2-saddle, and maximum components combined.

Figure 35 shows results on two different seismic fault datasets. These datasets are the output of a level set fault computation (section 4.2), which is represented as a signed distance transform. The first dataset Seismic-A contains a relatively sparse arrangement of faults with a significant amount of noise. The second dataset in figure 7, Seismic-B contains a very large number of intersecting fault features which poses a great challenge to extract the medial surfaces and segment by hand. The approach successfully finds the medial surfaces of the fault data and is able to segment them into meaningful components. It should be noted that gaps appearing in the
original level set distance transform remain in the segmented medial-surface, which is the desired behavior. Figure 36 focuses in on 2 different fault regions from *Seismic-A* to show medial surfaces extracted as 1-saddles, 2-saddles, and maximums in a display that is less complex than looking at the entire output.

This section presented a new technique that performs well on both synthetic and noisy real-world 3-D objects that contain medial surfaces. As an extension to this work, it would be interesting to use second-order curvature information in addition to the structure tensor for detecting more continuous medial surfaces. Later in section 4.5, the segmented medial surfaces presented in this section will be used to feed back into a level set process, which will then evolve according to localized information contained in its segmented medial surface.

![Figure 35: Medial-surface extraction and segmentation results from two different seismic datasets. Top row shows *Seismic-A* and bottom row shows *Seismic-B* as (a,e): original level set simulation output, (b,f): level set distance transform, (c,g): medial surface slices, and (d,h): segmented components.](image.png)
Figure 36: Analysis of results on Seismic-A fault dataset as (left): 1-saddle and 2-saddle medial surfaces and (right) a maximum medial surface. Arrows approximate the local eigenstructure at the critical points.

4.4. **Externally Generated Motion in the Normal Direction**

In sections 4.1 and 4.2 techniques are described for evolving implicit surfaces according to input data attributes and later in section 4.5 a technique is presented for evolving a surface according to its topology, but there still exist other pieces of information that can be exploited for surface evolution. For instance, stratigraphic features like channels (section 2.2.4) always extend further horizontally through a volume as compared to vertical motion. The opposite is true for faults (section 2.2.3) where motion in the vertical direction and horizontal direction parallel to the surface is much greater than movement in the horizontal direction perpendicular to the fault surface. This observation can be generalized as global knowledge about the nature of features, and this section describes how to integrate that information into the level set equation using externally generated motion.
Surface motion in the normal direction was first described in section 2.4.3 and motion based on an advection field for growth in the medial-surface direction is later described in section 4.5. The purpose for this section is to use domain knowledge to create an external velocity field that constrains surface motion occurring in the normal direction along a preferred orientation. This will become the basis for the medial-surface guided growth where an advection field is generated based on the orientation of the medial-surface near the level set front.

The basic equation describing the evolution of an implicit surface based only on an externally generated field is given by the advection (or convection) term that was contained in equation Equation 2-13. This representation of an advection field represents the deformation of an
implicit surface based on a given direction. Recall (section 2.3.2) that there is an important distinction between a direction and an orientation, where an orientation implies $\pi$-periodic behavior such that in 2-D orientations are limited between $[0, \pi)$ instead of $[0, 2\pi)$. This is achieved by defining an external velocity field that points in one direction of the desired orientation-based growth. Next, this growth is constrained to the surface normal direction by computing the dot product of the advection field with the surface normal. Outward orientation growth is then accomplished by using the absolute value of the dot product such that vectors pointing in opposite directions, which is the same orientation, achieve the same growth as vectors pointing in the same direction. Additional modifications can be made to this term by wrapping the dot product with a tapering function in order to create non-linear drop offs of the term as the normal diverges from the orientation field. The derivation for this term based on the basic level set equation (Equation 2-6) is given below:

$$
\frac{\partial \phi}{\partial t} = -\nabla \phi \left[ \vec{A} \cdot \vec{N} \right] = -\left| \nabla \phi \left( \vec{A} \cdot \frac{\nabla \phi}{|\nabla \phi|} \right) \right| \quad \text{Equation 4-6}
$$

where $\vec{N}$ is the normal direction and $\vec{A}$ describes an advection field at every point $i$ in the level set domain $\phi$.

The effects of this representation can be immediately understood by considering the situation where $\vec{A}=<1,0,0>$, which causes the implicit surface to stretch horizontally along the $X$-axis as shown for the box shape in Figure 37. Using an advection field where $\vec{A}=<1,1,1>$, results in a general expansion of shapes such as the spherical object shown in Figure 38. Although these deformations are visually appealing, it is important to point out that they are not growing limbs or conducting inverse curvature flow (section 4.5) since the growth field is defined independent of the model and does not consider an object’s topology. The application of this technique is to combine it with other approaches such as growing faults or channels where preferred orientations exist and can be defined externally of the object.
4.5. **Medial Surface-Guided Shape Exaggeration**

The planar level sets presented in section 4.2 can represent a unique set of objects, but the deformations that can be made to these surfaces are somewhat limited. In particular, it is desired to encourage outward growth in high positive curvature regions of the surface, such that planar features can extend further in their planar direction. This is desired behavior when segmenting geologic features like faults and channels, so that the bounds of a surface can be extended past the extents of an initial segmentation in order to fully encompass a feature. As described in section 2.4.5, this results in solving the ill posed backwards heat equation and is a non-trivial exercise. It should be observed that information provided by the segmented medial surface of a planar level set as described in section 4.3 could be leveraged to encourage growth along the medial surface, which would result in extending high curvature regions. Essentially, medial surfaces contain compact orientation information that points in the direction of high-curvature regions of a shape. Using this as a motivation, this section describes an approach for growing high-positive curvature regions of implicit surfaces by guiding motion along the object’s medial surface and allowing restrained evolution of an inverse curvature flow. This allows limbs to grow in a stable fashion since oscillations on the front remain smoothed by a restrained curvature motion. Background information on solving inverse curvature flow (via the backwards heat equation) was presented in section 2.4.5. A large portion of the work done in this section was presented at the 2008 IASTED Visualization, Imaging, and Image Processing [67].
4.5.1. Introduction

Growing limbs or regions of high positive curvature outwards on an implicit surface is similar to solving the ill posed and numerically unstable backwards heat equation. This can be interpreted as the opposite of mean curvature flow [121], which is a desired behavior for many applications of shape description [86], segmentation, caricaturization [128], and object modeling.

In a variety of physical phenomena, one wants to track the motion of a front whose speed depends on the local curvature. Two well-known examples are crystal growth and flame
propagation, but more general examples exist in the area of surface deformation. The basic approach to conduct the growing of limbs requires applying a local smoothing operator in the reverse direction (see Figure 39), which is an inherently unstable process. The technique presented here does not solve inverse curvature flow, but it allows the growing of high-positive curvature regions of implicit surfaces by guiding motion along the object’s medial surface and restraining the evolution to the initial surface, which allows limbs to grow in a more stable fashion. This results in desirable deformations of implicit surfaces. The algorithm is demonstrated on a number of generic shapes with well-defined skeletons or medial surfaces that contain sheet-like surfaces. Finally the technique is applied to extending planar geologic fault surfaces to be used for seismic data interpretation.

Curvature flow causes an interface to move inwards in its normal direction according to curvature. As analyzed in 2-D by Gage et al. [42], Grayson et al. [44], and Huisken et al. [55], convex parts of the interface move inwards while concave sections move outwards. Curvature flow smoothes out oscillations on the interface, which reduces noise. For growing limbs, the desire is to invert this flow such that convex regions move outwards and concave parts stay fixed or move inwards. This is similar to the problem of growing a sphere where Mullins and Sekerka [93] showed that it was an unstable process due to growing perturbations on the boundary; therefore this is a non-trivial problem. In this work, methods are presented for controlling the propagation of 3-D shapes for growing limbs such that some stability is achieved.

A number of techniques have attempted to grow limbs for two-dimensional problems. In particular, a significant amount of work has been conducted in the area of crystal growth and dendritic solidification, which simulate the physics of the development of fingered growth in crystals by a technique that links interface motion to a heat diffusion equation [122]. In this work on crystal growth, anisotropic motion was directed along symmetric axes for a preferred growth direction, which can generate large limbs. Unfortunately, this growth process is naturally unstable since small perturbations of the initial data can produce large changes in the result. This
anisotropic growth is expanded upon in [86] where growth is extended along the vertical and horizontal (x- and y-) axes for enhancing 2-D alphanumeric characters. Increased stability is achieved in this approach by shrinking limbs under curvature flow for a number of time steps, then using the shrunken object and the initial object, interpolating backwards in time to a previous state. This process produces oscillations when the time step is refined and therefore calculations must be done on a coarse scale for only a couple iterations in order to successfully extend protrusions. In an expanded technique, Steiner et al. [128] developed a restrained version of reverse mean curvature flow that allowed enhancement and caricaturization effects on 2-D features. This technique yielded an enhancement process that converged to a somewhat steady state. The technique presented here expands on the work just described by introducing a level set evolution that exaggerates and grows limbs in high-positive curvature regions for complex three-dimensional shapes. Utilizing the medial-surface or skeleton of an object, anisotropic growth need not be restricted to along arbitrary coordinate axes as in [122] but can instead be directed along the true orientation of the shape.

Some explicit techniques already exist for deforming objects such that regions of high positive curvature can grow outward. A discrete form of 2-D shape exaggeration is given in [128] where the discrete smoothing transform [18][121] is reversed using a restrained evolution. A steady state can be achieved (in some models) by using restraining forces similar to what is done in the deformable templates relaxation approach [73][146]. In this work the focus is only on implicit representations of shapes and therefore the explicit case is not considered.

In the following, a technique is described for growing high-positive curvature regions of implicit surfaces by guiding motion along the object’s medial surface and allowing restrained evolution of an inverse curvature flow. This allows limbs to grow in a stable fashion since oscillations on the front remain smoothed by a restrained curvature motion. Medial surfaces contain compact orientation information that points in the direction of high-curvature regions of a
shape. This orientation information is used to create a flow field that grows the implicit surface along the medial-surface and outwards at regions of high-positive curvature.

Figure 40: Example of evolution becoming unsteady for inverse curvature flow.

4.5.2. Restrained Inverse Curvature Flow

The process begins preventing discontinuities from appearing on the front during inverse curvature flow by normalizing curvature values. Since the radius of curvature is linearly related to the distance of level set (see Figure 41), the curvature can be approximated at each point on the level set front \( \phi = l \) at level \( l \) to the value of \( \phi = 0 \) by extending the curvature of the initial front to the nearest points in the domain. This could be accomplished by using the fast marching method [123] to propagate the curvature term on the initial front throughout the level set. A simpler approach is presented in Steiner et al. [128] where the curvature \( \kappa \) can be set to be:

\[
\kappa(t)' = \left( \frac{1}{\kappa(t) - \phi(t)} \right)^{-1}
\]

Equation 4-7
which approximates the curvature at each point to the curvature of its closest point on the initial front, without having to compute the distance transform. \( \kappa' \) (prime) designates the approximation of curvature \( \kappa \). Results from this flow can be seen in Figure 40 as the evolution becomes unsteady after about 20 iterations. This curvature value can be further adjusted by limiting values of \( \kappa \) to \( 0.0 \leq \kappa \leq 2.0 \), such that negative values of curvature are set to zero and the maximum allowable positive curvature is 2.0. This is similar to an inverse Min/Max flow [85] where only Max-flow is used and clamped for high values. A maximum allowable curvature value of 2.0 is used since in a three-dimensional grid this relates to a curvature radius of half a voxel \((\kappa = 1/R)\). To allow this the term \( H \) is introduced, which clamps the curvature \( \kappa \) between \([0,2]\).

\[
H(\kappa') = \begin{cases} 
0 & \kappa' \leq 0 \\
\kappa' & 0 < \kappa' < 2 \\
2 & \kappa' \geq 2 
\end{cases}
\]

Equation 4-8

The surface continues to evolve by propagating the level set for a short time step \( \Delta t \), then re-computing the distance function that represents the level set domain. A new value of \( \kappa \) can then be re-computed as the propagation proceeds using the modified inverse curvature flow.
Figure 41: Approximating the curvature at each point ($\phi=1$) in the level set domain to the curvature of its closest point on the initial front ($\phi=0$). Solid black curve represents the zero-valued level set and dashed black curves represent the level set at different distances. Since the radius of curvature (red circles) is linearly related to the distance of level set values ($\phi=dr$), the curvature at the initial front can be assigned to nearest points in the domain (orange arrows). Radius of curvature is $r$ for $\phi=0$ and $r+\phi$ for $\phi\neq0$.

Next, the evolving surface is restrained to the initial shape in order to allow the evolution to reach a more steady state. Applying the modified inverse curvature flow for infinite time causes the surface to expand into infinity and therefore a way is needed to achieve a more stable evolution. In explicit terms, this can be thought of as using imaginary strings to pull each point on the surface back to its initial location. Implicitly this is accomplished by creating an attraction force between the original and evolved shape [128] that restrains the evolution as in:

$$\phi = -\kappa'(x,y,z;0) \cdot |\nabla\phi| - \alpha \cdot \frac{(\phi(x,y,z;t) - \phi(x,y,z;0))}{\kappa'(x,y,z;t)}.$$  

Equation 4-9

The first term in the constrained evolution equation is the clamped basic equation for curvature flow defined in Equation 4-7. The second term defines an attractive force proportional to the
deviation of the evolving surface from the initial shape. The attraction works such that greater attraction forces are applied to regions that evolve rapidly. The result is high-curvature regions that evolve quickly away from the initial shape are held back. The parameter $a$ acts as a restraining factor where large values of $a$ cause the front to slowly deviate from the initial shape and become restrained. As $\alpha$ approaches zero, the attraction term vanishes and the evolution proceeds unrestrained.

Figure 42: Description of medial-surface anisotropic flow for a 2-D star, such that shapes deform and extend in the direction of their local medial surface (solid colored arrows).

### 4.5.3. Medial-Surface Anisotropic Flow

A technique is now described for using the medial-surface to guide inverse curvature flow and shape exaggeration (Figure 42). The level set method tracks the motion of an interface by embedding the interface as the zero level set of a signed distance transform, which must be periodically reinitialized using a technique such as the fast marching method. The signed distance transform is negative inside the level set surface and positive elsewhere. Therefore when locating the medial surface, a restriction can be made to look only where the distance transform is negative, since this represents the inside of the level set.
Recall that a technique for extracting the medial surface of implicit objects was already described in section 4.3. This technique provides a simple and compact representation of an object that preserves many of the topological and size characteristics of the original. Since the medial-surface of an object contains the local orientation of an evolving surface, this information can be used to guide the flow for growing limbs. The normal of the medial surface is extracted and compared with the normal of points on the level set in order to derive a strength of the directional flow. Representing the normal vector of the medial surface as $\mathbf{N}_{\text{med}}$ and the normal vector of a point on the level set as $\mathbf{N}_{\text{xyz}}$ the following equation can be defined for directed anisotropic flow:

$$F = \cos \left( \frac{\pi}{2} \cdot \left( \mathbf{N}_{\text{med}} \cdot \mathbf{N}_{\text{xyz}} \right) \right),$$

Equation 4-10

which combines into the following level set equation:

$$\frac{d\phi}{dt} = -\nabla \phi \left( F(H(\kappa')) \right) - \alpha \left( \phi(t) - \phi(0) \right) \frac{H(\kappa')}{H(\kappa')}.$$

Equation 4-11

The dot of the two normal vectors is a value between (0,1) which is inverse scaled to (1,0) using a cosine taper.

Using the dot product of the two normals, a force is achieved that creates a situation where when the normal of the medial surface is perpendicular to the level set front forward growth is encouraged. In the situation where the normal of the medial surface and the level set front are nearly parallel, growth is discouraged since this would not allow for the growing of limbs. This creates the same effect as the motion presented in section 4.4.
4.5.4. Results of Combined Flow for Restrained Exaggeration in 3-D

Results are now presented for the medial-surface guided level set exaggeration on a number of generic 3-D shapes to demonstrate the technique. The results show the initial shape, the extracted medial surface, and the exaggerated shape for a number of iterations. It can be seen that regions in the initial shape where high-positive curvature is present results in the growing of limbs such that convex regions extend outwards and concave regions stay mostly constant. It
should be obvious that significant shape exaggeration is achieved in these results, and for lower numbers of iterations the results are acceptable.

Figure 44: Medial-surface guided curvature flow for 3-D cross object. Left to right, top to bottom, medial surfaces of object followed by iterations 0, 10, 20, 30, and 40 respectively.

Figure 45: Medial-surface guided curvature flow for 3D linear object. Left to right, top to bottom, medial surfaces of object followed by iterations 0, 10, 20, 30, 40 respectively.
Figure 46: Medial-surface guided curvature flow for 3-D gear object. Left to right, top to bottom, medial surfaces of object followed by iterations 0, 10, 20, 30, and 40 respectively.

Although, these results still show some signs of instability after 30-40 iterations (see Figure 44, Figure 45, and Figure 46), when the restraining term is relaxed in the level set evolution. This suggests that further constraints need to be included in the approach in order to ensure more stable general flow. Closer examination of the 3-D objects computed with this technique shows an embossing effect is made on finer details of the objects that were exaggerated, an expected result of the inverse curvature flow terms. In Figure 47 the technique is applied to the results of a geologic fault simulation resulting from work done in section 4.2. Geoscientists typically desire fault surfaces resulting from simulations to be extended in their planar direction, and this technique is able to accomplish that goal.
This section presented the first technique for exaggerating and growing limbs of high positive curvature regions outwards in 3-D objects. This technique will find use in many areas of shape description, segmentation, and deformable surfaces. In a particular application area, it has been shown that this technique is useful for representing the planar surfaces of geologic faults in seismic datasets such that the extents of the fault surface can be extended. Additionally, applying techniques in this work to exaggerate protruding features such as noses and chins can create three-dimensional caricatures of people and animals useful for the entertainment industry. Future work will be undertaken for allowing this technique to converge to a true steady-state solution and to further improve on the stability after large deformations take place.

### 4.6. Multiple Region Flow

In chapter 4.1 and 4.2 techniques were presented for representing 3-D geologic features as a set of connected voxels in seismic data. These techniques used level sets to segment channel and fault features from seismic datasets by using a specific velocity function (see 2.4.1) of the level set adapted towards the features of interest. The possibility of using multiple attributes and seismic facies to represent other geologic features, such as for use in reservoir modeling, was
described in section 3.5. This section will focus on integrating multiple attributes of seismic datasets in order to drive level sets towards geologic features in a general representation.

Reservoir bodies are typically delimited vertically by top and bottom horizons and also delimited laterally by either the intersection of these two horizons or by faults. These boundaries enclose a volume that an interpreter would like to extract. Unfortunately, these bounding surfaces may not always be geometrically “sealing” and therefore techniques are desired to estimate reservoir volume without leaking out of approximate boundaries. Simple flood-fill techniques will not work in this case and leak out of any gap, but an evolving implicit surface can provide a robust solution to segmenting the reservoir. An additional advantage of segmenting a reservoir using level sets is that it is represented as an implicit surface, which allows for the generation of regularly spaced gridding and segmentation of complex reservoir topologies.

Velocity functions are defined that are adapted to the extraction of reservoir bodies from within bounding surfaces. Three-dimensional seismic facies (section 3.5) can help characterizing the reservoir. These facies represent typical geological features, which in turn can be linked to lithologies, thereby giving a more detailed and better structural description of the reservoir. Level set velocity functions can be based on characteristics of seismic facies in addition to the standard parameters of porosity and permeability used in reservoir modeling. Additional attributes can be derived from a data assimilation approach, such as growing neural gas [79]. Assuming multiple attributes of known characteristic, a level set velocity function is defined based on an integration of these quantities.

In order to accomplish reservoir body segmentation, it is assumed that multiple regions have been identified using a technique such as watershedding (section 3.5). Next, it is up to the interpreter to assign a unique velocity function to each watershed region. Since these watershed regions are created based on the grayscale values of some preliminary attribute, a simple grayscale threshold can be defined to each of these regions in order to precisely define the flow in
that region. By defining such a threshold and applying that to a thresholded velocity function (Figure 26), the level set evolution will proceed according to multiple velocity functions.

An additional consideration needs to be made in order to prevent edge effects from occurring along the watersheds that define regions of the volume. If multiple velocity functions are used in discrete regions, watershed edges will be visible in the output segmentation due to a sharp transition in the velocity function along these edges. In order to soften this transition, a simple volume blending approach can be used to create a smooth representation of the global velocity function across regions. This can be implemented by calculating the distance transform along the boundary of each discrete region and throughout the remainder of the volume. When this is accomplished for every discrete region, a new velocity value can be recomputed at every point in the volume based on a Gaussian blending (section 2.3.1) of the nearest regions to the point under consideration. Blending in this way works such that voxels in the center of a region receive all of their velocity from that region, but when moving closer to the edge of a region the influence of neighboring regions’ velocity functions become smoothly blended into a new velocity at that point. The size of the Gaussian smoothing operator can be adjusted (Appendix B.2) in order to provide more or less blending at region boundaries. The result of this combined approach is a technique for integrating velocity functions based on attributes in multiple regions into a unified model that can be applied to seismic reservoir modeling [73] and segmentation.
5. Interactive “Visulation” Environment (IVE)

This chapter implements techniques described in chapters 1 and 4 for the interactive visualization and simulation (visulation) of implicit surfaces for interpreting seismic data. First, an algorithm is described that will be implemented for solving level set surface evolution on a modern graphics processing unit (GPU). Next, a technique will be presented for real-time visualization of the implicit surface as it is evolved on the GPU. After describing the real-time visulation, interactive techniques will be presented for generating seed inputs to the level set along with a novel way for knowledge-based merging of discrete surfaces interactively. Next, an interactive technique is presented for editing and steering level sets in real-time by localized techniques that edit the velocity function. This chapter concludes with a user study to quantify the advantages of interpreting geologic surfaces using these techniques.

5.1. Level Set “Visulation” on the GPU

5.1.1. Introduction to GPGPU

Significant increases in the performance of GPUs along with recent improvements in their programmability has generated a new field of research focused on allowing general-purpose computations to be conducted on graphics processors (called GPGPU). Mainstream CPUs, such as those by Intel and AMD, have driven performance in computational applications for decades, yet in recent years the floating-point performance of GPUs has increased dramatically over CPUs (see Table 2) [101]. In particular, the peak throughput of an Nvidia GeForce GTX 295 (see Figure 48) is 1788 GFlop/sec, while a high-end CPU such as the quad-core Intel i7 Extreme 965 has a significantly lower peak theoretical throughput of around 51 GFlop/sec. Therefore, significant computational performance can be achieved by solving problems using the GPU.
Table 2: The peak throughput (measured by floating-point operations) of programmable GPUs has increased considerably over multi-core CPUs in recent years [28]. Peak performance on the Nvidia GeForce GTX 295 is 1788.48 GFlop/sec, while a quad-core Intel i7 Extreme 965 has a significantly lower peak theoretical throughput of around 51 GFlop/sec.

Why is the performance of GPUs so much greater than that of CPUs? The difference lies in the fundamental architectures for each device. Due to the parallel nature of computing graphics, GPUs can dedicate a majority of their transistors for arithmetic computation. CPUs, on the other hand, are optimized for sequential codes where transistors must be assigned to non-arithmetic tasks like branching. Therefore, in order to achieve performance anywhere close to the theoretical peak on a GPU, problems must be adapted to the highly parallel architecture of a GPU. In particular, these problems should offer a high arithmetic intensity, which is the ratio of arithmetic operations to memory operations. Memory latency is therefore hidden by computation, instead of the approach CPUs take by using large on-chip caches. If problems cannot be effectively adapted to take advantage of the GPU’s highly parallel compute capabilities, performance will likely degrade to the point where there is no advantage to using the GPU.

5.1.2. CUDA Overview

As recently as the year 2000, it was not possible to write programs on the GPU. However in the years since then, more and more programmability has been added to newer generations of
GPUs starting with allowing vertex-level programming and soon followed by pixel-level programming. During this time, knowledge in graphics languages (i.e., OpenGL) was necessary to program on the GPU due to the lack of a general API for the devices. In addition, early programmable GPUs lacked the ability to conduct scatter operations as well as having no support for 32-bit floating-point precision, which limited their application to many scientific problems. In 2007 Nvidia released a new programming paradigm called CUDA (Compute Unified Device Architecture) [95] in order to address many of these concerns.

CUDA is a hardware and software architecture designed for issuing and managing general purpose computations on the GPU. This allows the GPU to be treated as a data-parallel supercomputer with no need to write functions within the restrictions of a graphics language. CUDA is designed for use on Nvidia’s GeForce 8 series and newer models of GPUs, and allows advanced capabilities such that GPUs can be programmed more like regular CPUs. Some of the advantages gained with CUDA include scatter operations, 32-bit floating-point precision, high-speed shared memory, and full-support for conditionals and branching. An attempt was made by the ATI graphics company to produce a similar interface for their devices known as CTM (Close To Metal) [3] that later became known as Stream SDK, although this system never gained wide acceptance. Going forward, a consortium is working on generating an open standard (OpenCL) [75] that will allow CUDA-like functionality for a wide variety of platforms including those by Nvidia, AMD, and IBM (Cell). Until OpenCL is available, CUDA is the best platform upon which to implement GPGPU applications.

The CUDA programming interface is an extension to the C programming language and therefore provides a relatively easy learning curve for writing programs that will be executed on the device. CUDA programs follow a SIMD paradigm where a single instruction is executed many times in lockstep, but on different data and by different threads. In order to harness the full capabilities of CUDA, strategies and techniques are needed that utilize the architecture of CUDA-
enabled GPUs. Therefore, programmers need to understand these specific architectural details in a way that is not required for CPU programming.

Figure 48: Nvidia GeForce 295 GTX has a theoretical peak processing power of 1788 GFlop/sec. A GPU is installed to a PCI Express slot on a workstation.

5.1.3. Limitations with GPGPU

The reason CPUs haven’t been entirely replaced by high-performance GPUs is because there are limitations on the types of problems GPUs can efficiently solve. Alas, the GPU is not a universal accelerator for any application. For instance, the algorithm for clustering and classifying medial-surface components as described in section 4.3.2 will perform poorly on the GPU due to the difficulty in parallelizing random access to a global heap structure. In general, techniques that require maintaining global data structures where non-local (i.e., random) information is frequently accessed are not suited for the GPU.

The most computationally expensive component of the system presented in this thesis is easily the “visulation” of the implicit surface evolution. Over 98% of computation time is spent in this stage of the system. Therefore, efforts in this chapter are focused on parallelizing and accelerating this portion of the system using the GPU. Since methods like seeding, medial surface extraction, clustering, merging, occur before or after surface evolution and are primarily sequential, it is not advantageous to implement them on the GPU [4].
GPGPU is often compared to computing on the IBM Cell processor since it is another popular microprocessor designed to accelerate computations and be used alongside a regular CPU. The Cell processor provides a more general programming model than CUDA, such as allowing one group of threads to produce data while a different group of threads to consumes it. Although, CUDA provides an easier programming model to assign a computation to thousands of threads in a way that would take significantly more effort with the CELL. In terms of memory, applications on the CELL can tackle bigger problems more efficiently since its primary data storage is the host’s main memory. GPUs, on the other hand, have to transfer data from host memory to device memory for solving large problems. Fortunately, when data can be made to fit entirely in device memory, GPUs can access it with higher bandwidth than the CELL.

5.1.4. Streaming Level Set Algorithm

Level set surface “visulation”, first proposed by Lefohn [82], is a task well suited to being computed on the GPU due to the dense volumetric representation of the level sets and the localized finite differencing used to calculate derivatives. The level set algorithm developed to compute the implicit surface visulation will be described in the context of stream processing, which is a SIMD model of parallel processing described by a data set (stream) and an operation applied to the stream (kernel function) [21]. This model of processing is suitable for applications that exhibit high compute intensity, data parallelism, and data locality, all of which are qualities of the level set “visulation” technique.

A streaming level set implementation requires three major components [82]: data packing, numerical computation, and visualization. The data packing focuses on optimally storing the 3-D level set function into GPU texture memory such that it can be accessed and indexed efficiently and is discussed in section 5.1.6. The numerical computation of the level set must be done in a way that takes advantage data locality and maximizes compute intensity of a kernel function during each iteration. The CUDA code for solving the system of equations used in the
level set evolution is included in Appendix B.7. The visualization component consists of a marching cubes kernel that extracts and displays the implicit surface at every iteration 5.1.7.

An initial seed point is used to start a level set segmentation and this seed point must be represented by its signed distance transform in order to enable level sets to be computed. A signed distance transform represents the arrival times of an initial front moving in its normal direction with constant speed, which is negative inside and positive outside of the initial front (see section 2.4.2). As mentioned previously, this distance transform is most often computed on the CPU using the fast marching method [124], which maintains a heap data structure to ensure correct ordering of point updates. Unfortunately, this technique does not map well to a streaming kernel due to the trouble of maintaining the heap structure on a GPU. Therefore an iterative method is used to allow the distance transform to be computed in-stream.

The fast iterative method (FIM), by Jeong et al. [62], calculates the distance transform used for initializing the level set front. The FIM is an appropriate technique for streaming architectures, like GPUs, due to the way local and synchronous updates allow for better cache coherency and scalability. FIM works by managing a list of active blocks that are iteratively updated until convergence is reached. A convergence measure is used to determine whether or not blocks should be added or removed from the active list through synchronous tile updates. The reader is referred to [62] for further details and pseudo code describing this technique. A modified form of this technique was implemented for the CPU and resulted in similar compute times compared to the heap-based fast marching method. The resulting distance transform was identical for both techniques, which is the expected result. Therefore, it is valid to compare solution times between the CPU and GPU distance transforms.

5.1.5. Thread Decomposition

In the CUDA paradigm, the threads that execute a kernel are organized as a grid of blocks. A block is a batch of threads that work together and communicate by sharing data
through the local shared memory and can synchronize their memory accesses. Threads in different blocks cannot communicate or synchronize with each other. At the lowest level, a warp is a sub-set of threads from a block that gets processed at the same time by the streaming microprocessor. Warps are issued in an undefined order by a thread scheduler and therefore cannot be synchronized, so the lowest level of thread synchronization occurs at the block-level. This block-independence is what allows the CUDA architecture to scale well since as more processing units are added to future devices, more blocks can be independently computed in parallel.

A block-based updating scheme is used during computation on the IVE such that a block of threads share resources and work in parallel to update blocks of the solution. Similar to the approach used in [62], blocks are fixed to a size of 8x8x4 such that 256 threads are executed in parallel and have access to same region of the volume stored in shared-memory. A one-to-one mapping of threads to voxels is used in this implementation, such that a block of 256 threads computes the solution iteratively for blocks of 256 voxels until the entire grid of all voxels have been computed. For a grid size of $256^3$ voxels it takes approximately $256^2$ individual block updates to compute a solution.
5.1.6. Memory Decomposition

A 3-D CUDA array mapped to a texture is used to represent a volume on the GPU. The data is stored in 32-bit floating-point for both the input volumes and the level set volumes. It is necessary to store the level set volumes in 32-bit floating-point to ensure accurate calculations. Depending on the application, as many as four input volumes can be necessary for representing scalar values that control level set terms. In addition, at least two level set volumes are allocated for conducting a ping-pong computation where the active and result storage volumes are swapped each iteration. Along with these volumes, three large texture-mapped CUDA arrays are allocated as look-up tables to implement the isosurface extraction routine for storing edges, triangles, and numbers of vertices. Finally, two vertex buffer objects (VBOs) are created for storing triangle vertices and normals used in rendering. It can be seen that a large amount of GPU memory is required in order to enable fast computation.
Figure 50: CUDA memory decomposition. 3-D volumes (circled in green) are stored on the device’s global memory (DRAM) as a 3-D texture. Blocks (yellow) of the volume are pre-loaded into the multiprocessor’s fast shared-memory before being computed. Each voxel of the volume is computed by a thread (ALU) in the multiprocessor and has on-chip access to data in the shared memory.

In order to most efficiently move data from global to shared memory on the GPU, it needs to be stored in global memory (DRAM) in a way that allows reads to be coalesced. Coalesced memory accesses by a streaming multiprocessor read consecutive global memory locations and create the best opportunity to maximize memory bandwidth. Therefore, packing a volume in global memory with the same traversing order as memory accesses made by the algorithm is the most efficient way to store a volume in global memory. This can be accomplished in a straightforward manner by re-ordering a volume such that 8x8x4 blocks of the volumes occur consecutively in linear memory. Next, the re-ordered volumes in global memory can be mapped to CUDA textures, which provides an opportunity for data to be entered in a local on-chip cache (8 KB) with significantly lower latency. With this combined approach, chances are
significantly increased that when data is requested from global memory it will be cached either from texture mapping or when requesting nearby memory locations. Memory reads are thereby optimized as long as there is some locality in the fetches. For the purposes of this work, non-local texture fetches rarely need to be made since the level set computation requires access only to neighboring voxels in the volume. In practice, texture memory that has been cached can be accessed like an L1 cache (1-2 cycles) as compared to global (non-coalesced) memory reads that require a significant 400-600-cycle latency. In practice, these numbers will vary greatly depending on exact memory access patterns and how often the texture cache needs to be updated, which cannot currently be controlled by the programmer using CUDA.

There are significant advantages to reading from texture memory as compared to global GPU memory, which is necessary to experience the full benefits of the GPU architecture. Textures act as low-latency caches and provide higher bandwidth for reading and processing data. In particular, 3-D textures are optimized for 3-D spatial locality such that localized accesses to texture memory is cached on-chip. Textures also provide for linear interpolation of voxel values through texture filtering that simplifies rendering at sub-voxel precision (see Figure 52). Data access using textures also provides automatic handling for out of bounds addressing conditions by automatically clamping accesses to the extents of a volume.

Since shared memory provides over 2 orders of magnitude faster access to data than global memory, it should be pre-loaded with data that is expected to be frequently used. Shared memory is first set aside for the storing the level set volume, since it is the volume most frequently accessed during computation of the evolving surface (e.g., when calculating finite differences). Blocks of size 8x8x4 comprise 256 floating-point values or 1KB of shared memory. Since each SM has 16KB of shared memory available, additional data can be stored in the remaining memory. This memory should next be assigned to the bordering voxels around each block (~1KB) so that level set values computed on block edges do not have to access global
memory. Next, blocks of size 1KB can be stored from any of the feature volumes that provide information for the level set terms such as the input seismic data or a fault likelihood volume.

Since shared memory, caches, and registers are shared by one or more blocks on a SM, there is a limit to how many blocks can be launched at one time. This depends on how many SM resources are required by a single block. Therefore, there exists a tradeoff between block sizes and the use of shared memory since larger block sizes will require more data to be accessed from shared memory, thereby reducing the amount of data that can be stored in the banks. An additional concern is that the 16KB of shared memory is divided into 16 banks that allow for simultaneous access. When multiple banks are accessed at the same time (i.e., bank conflict [62]), requests must be serialized which causes performance to degrade. Block sizes of 8x8x4 provide a good middle ground between resource allocation per thread as well as being large enough to hide memory latency through many parallel computations.

### 5.1.7. Isosurface Extraction

One of the many advantages of using an implicit surface representation for modeling geologic features, as opposed to an explicit representation like a triangulated mesh, is its ability to dynamically adapt to drastically changing topologies and maintain a stable representation during computation. The disadvantage with the implicit representation is that it poses a challenge to extracting and directly visualizing isosurfaces of the function, something that comes cheaply with an explicit surface representation. The natural way to visualize an implicit surface is using direct volume rendering, which renders the implicit surface directly in its native state on the GPU. This could be accomplished by using a ray-marching pixel shader [119] to render the level set directly in the GPU texture. By marching rays through the volume and accumulating densities from the “3-D texture” as a stack of 2-D textures, a value for the level set can be rendered. Unfortunately, direct volume rendering is only a rendering technique and does not provide a final form of the surface that can be used for further processing and editing in a geologic model. More importantly
though, is that volume rendering is much more computationally expensive than extracting an isosurface to visualize using marching cubes. In order to assure that the speed of the IVE is as fast as possible, an isosurface extraction technique is used for visualization instead of volume rendering.

Figure 51: Extracting the triangulated surface from a voxel representation of the level set surface on the GPU and rendering it on the GPU.

Isosurface extraction using the marching cubes algorithm [84] extracts a triangulated mesh of the level set surface. This approach is desirable since the resulting surface is identical to the level set surface and can be used in the many mesh-based reservoir-modeling tools. For this reason, a new technique was implemented for extracting the isosurface of a level set surface using a modified streaming marching cubes algorithm that allows for fast and easy visualization on the GPU. Marching cubes is efficiently implemented to run on the GPU in a way that extracts triangles directly from the level set representation. This approach requires no further processing or intermediate storage of triangles prior to rendering and is therefore able to run at interactive rates (to be discussed in section 5.1.8). This method is an extension of prior GPU-based isosurface extraction techniques described by Johansson et al. [63] and Tatarchuk et al. [130]. The approach described here is unique in how isosurface triangles are immediately identified and classified when each level set voxel is computed as well as how the CUDA implementation is able to take advantage of shared memory when creating triangles.
The first step is to classify each voxel of the level set surface based on the number of triangle vertices it will generate (if any). In this voxel classification step, the goal is to determine whether each vertex of a voxel is inside or outside of the isosurface (i.e., level set) of interest. Next, the process iterates over all voxels in the volume and records the number of vertices that lie within the isosurface. If there are no vertices found for a voxel that lies within the isosurface, that voxel is designated as inactive so that it will be skipped.

The next step is to compact all active voxels into a single array. This is accomplished by iterating over the array that designates whether or not a voxel is active, and in the case where it is active, the voxel’s index is saved in a compacted array that contains only active voxels. In order to exploit parallelism during the isosurface extraction, a prefix sum (scan) is performed across the volume in order to determine which voxels contain vertices and compact those voxels into a single array, while ignoring empty ones with no vertices. This scan can be accomplished efficiently in parallel by using the prefix sum available in the CUDA Data Parallel Primitives Library (CUDPP) [49]. This scan results in a compacted array that ensures for the remaining steps the only voxels being calculated are truly active. Well-balanced parallelism is then accomplished by evenly dividing this compacted array among GPU stream processors.

The final step is to iterate over the compacted active voxel array and generate triangles for rendering. This is done by checking all active voxels in the compacted array and calculating the points of their intersecting triangles. Since the compacted array contains the locations of vertices where the isosurface intersected a given voxel, 3-D point locations are readily available. The three points that make up the triangle are then used to calculate the surface normal for the triangle face using a cross product. The triangle vertices and normal vector are then saved into vertex buffer objects, which are buffers on the GPU for storing geometric data. Finally, the isosurface is displayed by rendering the triangles in the vertex buffer. The normals are used for calculating the lighting and shading of the triangles. The result is a triangulated mesh.
representation of the implicit surface that is readily visualized on the GPU and can easily be transferred to main system memory for post-processing and editing at the end of a simulation.

Some optimizations to this technique allow isosurface triangles to be extracted immediately during the computation of a level set voxel. When a level set voxel is computed, it is straightforward to determine if that voxel lies on the isosurface of interest. If it is determined that the voxel lies on the isosurface, derivatives used in the calculation of the level set can be immediately re-used in order to determine the triangles at that point on the implicit surface. Using this approach, since the normal of the implicit surface has already been computed, it does not need to be recalculated and can be immediately assigned to the extracted triangles. This optimization significantly improves the speed at which surface components are extracted from the level set.

Figure 52: Tri-linear texture filtering on a seismic volume (top) and a level set volume (bottom). Left image is non-filtered and right image is filtered.
5.1.8. Results

The technique is implemented using CUDA version 2.0 on an Nvidia GeForce FX 5600 series graphics card and utilizes the CASI (Computer Aided Seismic Interpretation) development environment by TerraSpark Geosciences as the 3-D development platform (see Appendix C.1). The 5600 has 1.5 GB of global DRAM memory and 16 streaming multiprocessors (SMs) each of which contain 8 streaming processors (SPs). The SPs are clocked at 1.35 Ghz and each can performed a fused multiply-add (MADD) every clock cycle, which gives the device a theoretical peak performance of 345 GFlop/sec. Each SM has 16KB of shared memory that can be accessed by all SPs within the same SM. The 5600 is installed on a workstation with two AMD Opteron 280 dual-core processors at 2.4GHz. The Opteron contains three floating-point units per core and therefore has a theoretical peak performance of 2.4 GFlop/sec for a single core and a total of 28.8 GFlop/sec when using all four cores (theoretical peak increases to 7.2 GFlop/sec and 28.8GFlop/sec when considering all three floating-point units).

Data is moved from the CPU host memory to the GPU via DMA transfers across a PCI Express 1.0 bus, which has a peak bandwidth of 4.0GB/sec in each direction. Requests from the GPU to host memory are significantly more expensive than those accessing the GPU’s global memory, which has nearly twenty times the peak bandwidth at 76.8GB/sec. Since performance would certainly be dominated by the low bandwidth to host memory, this work uses data sets that fit entirely in the global memory of the GPU in order to work-around this bottleneck. Since the GPU used in this work has a respectable amount of memory at 1.5 GB and the latest devices have up to 4GB of memory, there is sufficient memory available on the GPU for tackling moderate-sized problems of 256³.

The technique is applied to three different subvolumes of a large seismic survey. Each subvolume contains a single geologic feature of interest: a fault (Figure 54), channel (Figure 53), and a high-amplitude geobody (Figure 59). The process is run for 200 iterations and times were
recorded for the CPU implementation and the GPU implementation as well as the number of triangle faces on the isosurface. Each iteration moves the surface at most a distance of 1-voxel from its initial position, and therefore solving for 200 iterations will drastically deform the initial seed points into the geologic feature; although it is a subjective decision to determine when sufficient growth has been achieved. The results for the CPU and GPU implementations were identical. The level set is computed each iteration for a narrow-band of 10 voxels around the feature. Whenever the surface evolves past the narrow band, the domain is re-initialized using the FIM (section 5.1.4). At the completion of every iteration, the isosurface of the zero-valued level set is extracted and immediately rendered to the display using the technique described in section 5.1.7.

Figure 53: Time series computed on the GPU (left to right, top to bottom) showing a channel surface evolving from a line of seed points.
Figure 54: Time series computed on the GPU (left to right, top to bottom) showing a fault surface evolving from a seed point in a seismic dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>CPU-1</th>
<th>CPU-4</th>
<th>GPU</th>
<th>Speedup</th>
<th>FPS</th>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geobody</td>
<td>128</td>
<td>225.35s</td>
<td>74.11s</td>
<td>13.44s</td>
<td><del>5</del>17x</td>
<td>14.9</td>
<td>216k</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1978.2s</td>
<td>625.0s</td>
<td>62.43</td>
<td>~11:32x</td>
<td>3.2</td>
<td>1729k</td>
</tr>
<tr>
<td>Fault</td>
<td>128</td>
<td>167.49s</td>
<td>57.75s</td>
<td>9.39s</td>
<td><del>6</del>18x</td>
<td>21.3</td>
<td>93k</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1382.5s</td>
<td>431.8s</td>
<td>38.24</td>
<td>~12:36x</td>
<td>5.23</td>
<td>744k</td>
</tr>
<tr>
<td>Channel</td>
<td>128</td>
<td>217.83s</td>
<td>71.47s</td>
<td>11.41s</td>
<td><del>6</del>19x</td>
<td>17.5</td>
<td>54k</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1816.7s</td>
<td>567.5s</td>
<td>49.06s</td>
<td>~12:37x</td>
<td>4.1</td>
<td>432k</td>
</tr>
</tbody>
</table>

Table 3: GPU versus CPU runtime for three datasets segmenting a geobody (Figure 59), a fault (Figure 54), and a channel (Figure 53) for 200 iterations. Speedup is approximated by the CPU times divided by GPU time. Frames per second (FPS) numbers are based on the GPU times. Faces are the number of triangles (in thousands) rendered as an isosurface. GPU times include data transfer to/from the CPU.
Table 3 shows the results of the technique comparing the GPU implementation to the CPU implementation on the three feature subvolumes. As can be seen in all three cases, the GPU implementation is significantly faster than the CPU implementation. For all three datasets, the process was 17-19 times faster on the GPU compared to the single-core CPU for the $128^3$ size and 32-40 times faster on the $256^3$ size. The GPU speedup was less pronounced when compared to using all 4-cores of the CPU, ranging from 5-6 times faster for the $128^3$ volume and 11-12 times faster for the $256^3$ volume. These results are encouraging and describe the advantages of using the GPU for interpreting geologic surfaces with level sets. The speedup should not only be considered by how much time the geoscientist saves waiting for results, but more importantly it allows observation of the segmentation in a way that is similar to watching a movie. As the segmentation process evolves on the GPU, it is being visualized at a rate of 14-21 frames per second (FPS) for the $128^3$ data and 3-5 FPS for the $256^3$ data. This speed is fast enough to engage

Figure 55: Graph showing speedup achieved with GPU implementation for the $256^3$ subvolumes in Table 3 evolved for 200 iterations. Speedup is 32-37 times faster on the GPU than on the CPU. CPU-1 is a single core and CPU-4 has four cores.
the geoscientist to stay involved and quickly understand the three-dimensional nature of their data as they see it evolving before their eyes. If the rate of FPS dropped below 1.0, it would be difficult for geoscientists to observe surface changes since the rate of change is too slow. Since CPU rates are far below one frame per second it is certainly not considered interactive.

<table>
<thead>
<tr>
<th></th>
<th>CPU-1</th>
<th>CPU-4</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Throughput (Rmax)</td>
<td>1.72 GFLOPS</td>
<td>5.49 GFLOPS</td>
<td>60.45 GFLOPS</td>
</tr>
<tr>
<td>Theoretical Peak (Rpeak)</td>
<td>2.4 GFLOPS (7.2)</td>
<td>9.6 GFLOPS (28.8)</td>
<td>388.8 GFLOPS</td>
</tr>
<tr>
<td>Percent Peak (Rmax/Rpeak)</td>
<td>71.7% (23.9 %)</td>
<td>57.2% (19.8 %)</td>
<td>15.5 %</td>
</tr>
</tbody>
</table>

Figure 56: Percent of peak performance calculated as Rmax/Rpeak from the 256³ tests for a single core of the Opteron (CPU-1), all 4 cores of the Opteron (CPU-4) and the GPU implementations. CPU-1 and CPU-4 results in parentheses are included to compare with previous works that use the three floating-point units in the Rpeak value.
Figure 57: Percentage of compute load and bottlenecks of the GPU implementation determined by using timers for the compute load in order to determine time spent in each function and by varying the memory and core bus speeds in order to identify hardware bottlenecks.

Figure 56 compares the performance throughput of each processor for the $256^3$ problem. The average throughput, also called $R_{\text{max}}$, is calculated as the number of floating point operations computed during each iteration multiplied by the number of iterations (200) divided by the simulation time (see Table 3). The single-core Opteron was able to achieve an average throughput of 1.72 GFlop/sec, the two dual-core Opteron achieved 5.49 GFlop/sec, and the GPU computed 60.45 GFlop/sec. When these numbers are divided by the theoretical peak throughput ($R_{\text{peak}}$) for each chip, we get a value for the percentage of peak performance attained. The single-core CPU had the greatest percentage of peak performance, which is to be expected since it didn’t need to manage parallel threads. The GPU had the lowest percentage of peak performance, which is also not surprising since attaining performance close to its theoretical peak is extremely difficult due to the number of parallel threads that must be kept busy for maximum throughput. It is encouraging, though, to see that the average throughput of the GPU (60.45 GFlop/sec) is over
six times the peak throughput the two dual-core CPUs. This means it is theoretically impossible to achieve the GPU’s performance with two dual-core CPUs.

Figure 57 looks at the individual components of the technique as well as the hardware components of the GPU in order to determine where bottlenecks exist. The compute load pie chart on the left looks at the three major components of the technique (i.e., distance transform, level set computation, and isosurface extraction) and shows how much time was spent in each stage during the course of the $256^3$ experiments. It can be seen that the majority of time (over 75%) was spent extracting and rendering the isosurface from the level set. This suggests that only limited speed improvements can be achieved through algorithmic changes for the distance transform and level set computation, but significant improvements can be made if the isosurface extraction is made faster. Ways to add further optimizations to the isosurface extraction were discussed at the end of section 5.1.7.

The graph on the right of Figure 57 compares the contribution of the GPU core speed and the GPU bus speed to the total GPU performance. In order to determine if the technique was limited by the core speed or the bus speed, an experiment was conducted by manually varying the core speed and bus speed for the GPU. This is sometimes referred to as overclocking and underclocking from the factory settings. Modern Nvidia GPUs allow for the clock and memory bus settings to be adjusted in software, and therefore it is straightforward to make these adjustments. The $256^3$ experiments from Table 3 were re-computed using both an underclocked core and an overclocked core, while keeping the bus speed constant. Next, the core clock was kept constant and the bus was underclocked and overclocked. Both the core and bus were changed by no more than 30% of the factory default speeds. By comparing the change in computation times for each of these situations, it was determined that when considering only these two factors the technique was 53% limited by the core speed and 47% limited by the bus speed of the GPU. These results show that by using a more modern GPU with a faster core speed
but the same bus speed, the allowable speedup will be significantly limited by the memory bus. Vice versa is also true.

There is a significant restriction to the problem size used in this experiment. A constraint was enforced to require that both the subvolumes and the overhead data from the technique fit entirely in 1.5 GB of GPU memory. Although this is a significant amount of GPU memory for today’s technology, it is still significantly less than the 32GB of memory available on workstations commonly used in seismic interpretation shops. The main memory sinks are the three to six 32-bit data volumes that need to be stored for representing two swappable level set volumes, a propagation volume, an advection volume, and possibly a volumetric representation of the starting seeds. The second largest memory sinks are the multiple VBOs that store voxel coordinates, vertices, and normals for the surface extraction. The VBOs are implemented as memory buffers that become active when bound, which essentially turn every pointer into an offset into GPU memory that can be shared between functions. This representation is powerful by allowing multiple bounds to a common buffer, allowing data in the buffers to be accessed externally. In general, for a problem of size $n^3$ many values are needed to be stored for each point in the volume, thereby creating a sizeable memory requirement.

Figure 58: Fixed-size bounding box interactive subvoluming of a large seismic survey.
Requiring all data to fit entirely in GPU memory limited the size of datasets to a significantly smaller scale than what is commonly interpreted on large workstations, such as $256^3$. Due to this size restriction, the example datasets were subvolumed to only contain a single geologic feature of interest. Although GPUs will certainly increase their memory capacity over time, it is expected that as datasets increase in size and resolution, there will always be a difference in the amount of memory available on a GPU compared to a workstation. Therefore, a paradigm has been established to approach interpreting large datasets by focusing on features in smaller subvolumes. A visualization application can easily allow for this type of dimension-restricted subvoluming by moving a fixed-size bounding box through a large survey (see Figure 58) and conducting GPU interpretation in a piecewise fashion to determine parameters to be used in a batch-style CPU cluster computation. The memory capacity of a GPU can obviously be circumvented by transferring data back and forth between the GPU and main system memory, but due to the limited bandwidth across the PCI Express bus, the possibility is not considered in this work.

Figure 59: Segmentation of high-amplitude geobody in a 3-D seismic volume showing (a) user defined seed point to start evolution. (b) and (c) show the extracted isosurface of the level set while it evolves at 50 and 200 iterations respectively.
5.1.9. Real-time Structure Analysis

In this section the prospect of real-time structure analysis conducted on the fly as the level set evolves is discussed. By conducting structure analysis on a narrow-band around the implicit surface in real-time, it allows geologic features to be imaged on the fly for driving surface evolution (see Figure 60). This can be a useful technique for analyzing data that cannot easily be pre-computed by standard structure analysis (Chapter 1), for adaptive segmentation of simple shapes, and for situations where input data is being received in real-time and must be immediately processed before it changes.

![Figure 60: Calculation of the structure tensor on the seismic data around a narrow-band of the level set returns propagation and advection terms on the fly for use in surface evolution.](image)

The techniques described in this work that use implicit surfaces to model geologic features require many level set terms (propagation, advection, etc) to be calculated before the implicit surfaces can be computed. The reason for this is largely due to computational efficiency, since it is more efficient to compute these terms all at once for the entire domain rather than on an as-needed basis. As described in section 5.1.1, GPGPU [47] is providing significant changes to
this paradigm by allowing for greater interaction and faster responses to data interrogations. The GPGPU paradigm provides sufficient computational power to calculate many level set terms on the fly in a way that steers the level set surface in real-time. This removes many of the requirements to pre-process the structure analysis of an input volume before it can be interpreted using level sets. This also provides geoscientists with a more immediate response to their interrogations. A future benefit to this work will be in fields where data is streaming in live from medical devices, supercomputer simulations, or computer-aided design and there is a need to quickly identify features as soon as the input data changes in real-time.

This real-time technique lets the user work directly from their original source data by identifying potential regions with seed points and employing a real-time structure analysis kernel to segment features while the user observes the surface evolve. Disadvantages with conducting real-time structure analysis are also identified such as the many parameters needed to be set and inefficient processing, and for these issues possible improvements are discussed.

5.1.9.1. Edge Detection of Basic Shapes

To set the stage for real-time techniques, a segmentation strategy is presented for use on basic shapes in order to demonstrate correctness, and then the following section focuses on seismic data features in order to show real-world applicability. Three basic geometric shapes that have been used elsewhere in this dissertation were again chosen to segment their edges in 3-D using a real-time kernel to drive surface evolution. The kernel used is a 3-D central difference edge detector that is calculated on the input data for every voxel being evolved. The result of the edge detector is then immediately used as the propagation term for the level set evolution, such that a high edge score corresponds to a greater propagation and a zero edge score keeps the surface stationary.

Figure 61 shows the results on one 3-D shape after the surface evolution reached a steady state. The segmentations were started with a single seed point chosen near the edge of a shape,
and then the algorithm segmented the remainder of the shape’s boundary. It should be stressed that the inside of these shapes remained hollow and only the inside and outside boundary of the shape is represented by the surface.

Figure 61: Real-time structure analysis of a 3-D shape as it is segmented using level set evolution. The level set uses only a propagation term that is based on edges in the volume.

5.1.9.2. Structure Kernels for Seismic Data

When applying this technique to seismic data features, the structure analysis techniques described in section 1 need to be written as a kernel. A kernel is a function called on a single voxel that returns some value based on the structural analysis of an input dataset and/or topological properties of an implicit surface. This value is intended to be used as a term in the level set evolution. In the previous section 5.1.9.1, the 3-D edge detector was a kernel used to generate a propagation term.

Although it seems straightforward to implement the structure analysis techniques as function calls for a given voxel, unfortunately this approach does not take advantage of the volumetric representation of the input data or the level set representation. Therefore many redundant calculations result. In particular, in the case where a kernel is being calculated on an input dataset that is constant and never changes, a single voxel may have its kernel calculated
multiple times. This is not a problem for a dataset where the ratio of feature to non-feature is very low, meaning the final surface will be small in relation to the size of the volume. In the case where the ratio of a feature to a non-feature is very large, such as if the final surface encompasses a large part of the input volume, this becomes a significant inefficiency. The reason is two-fold. First is the already mentioned fact that as the narrow-band of the level set computation moves through the volume, the same voxel will have it’s kernel calculated multiple times possibly resulting in redundant calculations. Second is that many of the structure analysis techniques require a series of calculations on a region of voxels around the voxel in the kernel. This regional computation becomes more pronounced in operations that conduct smoothing on a region of voxels around a center voxel, as the region of influence can be quite large. When a smoothing operation is cascaded by a subsequent computation that also requires a region of voxels, the previous operation must be first computed for each of the voxels in the region. The result is that for cascaded operations requiring a region of voxels, the kernel paradigm becomes far more computationally intensive than pre-calculating the level set terms at one time. Therefore, the kernels described in this section are adapted for simple structure analysis techniques and assume the input volume has been previously smoothed.

In order to calculate structural attributes of faults and channels in seismic data, the local horizon or stratum must first be found at a given location in the seismic data. This requires first calculating the structure tensor by finite differences and then finding the sorted eigenvalues and orthonormalized eigenvectors of the structure tensor. Since in the GPU implementation both the level set domain and the seismic data are stored in 3-D texture-mapped memory, memory values can be quickly retrieved for use in very fast derivative calculations for generating the structure tensor.

Next, the eigenvalues and eigenvectors of the structure tensor must be determined. This is accomplished by using a noniterative algorithm from the medical imaging community described in [50]. For solving the eigensystem, an algorithm was chosen that does not require iteration in
order to allow fast calculations of eigenvalues and eigenvectors that leverage the high computational throughput of CUDA. Iterative techniques can decrease the throughput of the GPU if they are not taking advantage of the large number of calculations that can be quickly computed on the GPU. Pseudo code is provided for this non-iterative solver in Appendix B.5.

After having a representation of the structure tensor and its eigenvalues and eigenvectors, it is straightforward to compute the kernels described for imaging faults and channels. The kernels are computed during each block update of the level set domain that was described in section 5.1.5. As every voxel in the evolving level set surface is solved, the feature kernel is first computed then the resulting values are immediately used in the level set equation. This order of computations is important because it results in feature kernels only being computed when the evolving surface is driven into that region of the volume. This also provides a layer of adaptivity to the technique since kernels can use information about the current position and shape of the implicit surface into the parameters and orientations of their computation.

5.2. Seeding Level Sets

All level set evolutions require a seed or set of seed points from which the evolution begins to grow. One standard way for accomplishing this is by using a shape-prior model, which approximates the object being segmented and helps the evolution proceed to a solution faster. Another more obvious way is by a manual seed, which is picked or drawn into the segmentation by the user. A number of techniques are described for creating seed inputs to level set evolutions for seismic interpretation applications. This section focuses on applications to fault segmentation, although the techniques described are generally applicable to other features.
Figure 62: Automatically extracted seed lineaments for seed points to the level set. Different colored lineaments represent distinct seeds that are approximated to align with faults in the data.

### 5.2.1. Automatic Seeds

Automatic seeds can be generated using techniques that approximate the location of features of interest. The goal of these techniques is that they are fast to compute and their approximation to the feature of interest is close enough to at least intersect at one point. In the case of geologic faults an automatic seed input can come from a lineament extraction technique that auto-tracks peaks [68] or from a traditional Hough transform operated on 2-D time slices of a 3-D volume. Both of these techniques attempt to trace features two-dimensionally (i.e., on horizontal slices) by following peaks in an attribute volume such as channelness (section 3.3) or a fault likelihood volume (section 3.4). Figure 62 shows an example of automatically extracted lineaments that approximate fault locations. The use of automatic seeds and peak-tracking techniques for identifying faults has been researched extensively and is currently available in commercial software. For the case of channels, automatic seeds are more difficult to generate due to the complexity of channel attribute volumes and the high-likelihood of false-positives. The same holds true for identifying other geobodies, which is why the seeding techniques in sections 5.2.2 and 5.2.3 are recommended for that purpose.
5.2.2. Semi-Automatic Refinement Seeds

Semi-automatic seeds consist of using the previous output of a level set evolution as the input to a new simulation. This approach can be used as a way to iteratively segment by using the output of a previous level set simulation as the input to a later simulation. This may be a desired order of operations if a user does not know how much evolution is necessary to segment a feature, and if not enough evolution was done previously so that the process can continue evolving further. Continuing a level set evolution that has reached its final iteration can be considered a special case of a semi-automatic seed. Figure 63 shows an example of a fault that was segmented using the planar level set approach (left) and afterwards using it as the input to a second level set evolution so that the gap can be filled-in, resulting in a better segmentation (right).

Figure 64: Manual Seeding of Level Sets for Planar Fault Extraction (white blocks represent manual seeds, green surface is the segmented fault).
5.2.3. Manual Seeds

Manual seeds are hand-drawn into the computer using an interaction technique similar to “painting”. This is the most versatile technique for creating seed points since it gives a user the most control over the process, but it also can be more time consuming than automatic and semi-automatic seeds. The manual seeding has been implemented in two ways by using either a cubic paintbrush that can be elongated in any direction or a point set dropper that places points at mouse cursor locations. In either case, the user moves along 2-D slices in the 3-D volume and places seeds at places that approximate the location of features. The result of the painting procedure is then used as the initial zero level set for segmentation. The different use cases between the two approaches are that the cubic paintbrush is typically used to enclose a feature of interest (as in Figure 64), whereby the surface shrinks and collapses around the feature with some outward growth. The point set dropper is used to define a sparse starting point that definitely intersects the feature of interest, thereby allowing the surface to grow extensively into the feature with minimal inward growth (see Figure 53, Figure 59, or Figure 54).

Figure 65: Example of the complex situation that results when interpreting thousands of discrete surfaces in 3-D. Surface patches become occluded and are difficult to discern.
5.3. **Surface Merging**

Section 4.3.2 described a technique for clustering planar features together based on their local orientation. In different application areas, it is not always possible to cluster or create discrete surface features that represent the characteristics of features in real-life. In these situations, most clustering techniques will fail and it is up to the user to determine the proper arrangement of feature characteristics. For this reason, a novel technique was developed for merging and breaking apart surfaces by using simple mouse strokes. Figure 65 shows an example of a large number of surfaces being visualized in TerraSpark Geoscience’s CASI software package. Combining any of these surfaces together is done in the 3-D render window by holding down a key and moving the mouse across the desired features. The following sub-sections describe further information on these merging and breaking techniques.

Figure 66: Manual merging of surface patches. White arrows describe the motion of a mouse cursor.
5.3.1. Manual Merging

Manual merging works by moving a mouse cursor through the 3-D render window across feature components that should be merged. As features are selected they become highlighted, which is visually accomplished by increasing the rendered size of all connected points (see Figure 66). When the user presses a key after having selected a number of similar features, they are merged into a single cluster, which is visualized by all components now having the same color. Before the merge finalizes and the features are actually combined, the user is able to visualize the final result in 3-D. If incorrect patches were selected to be combined, an additional keystroke cancels the merge process. Using this completely manual merging approach a user can quickly move through a moderate sized dataset and combine discrete surface patches into coherent features. Unfortunately for large datasets with many surface features, such as a heavily faulted dataset or a complex geobody, surface patches begin to occlude themselves and make it difficult to determine which patches should be combined.

5.3.2. Smart Merging

The use of local orientation information between features can help improve on the manual merging technique by identifying which surface patches are good candidates to be combined. This describes a new technique called smart merging. Two surfaces are often manually merged together if they have a common orientation and a close distance to each other. Therefore, using a smart merging technique would make it possible to pre-empt the merging decisions a user would make on their own.
Figure 67: Example of Smart Merging by selecting a patch for consideration (left) then highlighting all patches that meet the distance, normal dot product, and coplanarity constraints. (right) Highlighted points are then automatically merged into a new feature.

The smart merge works by when a surface patch is first selected for merging, a search is made for all other surface patches being displayed that are a given distance away. The distance is measured between two sets of points by using the distance of their midpoints. Although the midpoint approximation is not the most accurate way to compare the distance between two patches, it is fast and when the patches being used are compact it performs well. For those patches that are within the distance cutoff, their orientation is then compared to the first-selected patch. There are many ways to compare the orientation between two surface patches; the technique used here is to calculate the dot product of the normals and the coplanarity between the two patches. The dot product between the two normals is close to 1.0 when the normals are pointing in the same orientation. Coplanarity is calculated by taking the dot product of the first patch’s normal with the vector between the two midpoints of the patches. This dot product is close to 0.0 when the patches are coplanar. The results of these calculations are compared to three user-defined parameters: minimum distance, minimum normal dot product, and maximum coplanarity dot product. If the result passes each of these parameter tests, the current patch in the search queue is automatically merged with the selected patch.
Another way of thinking about the smart merging technique is as a lightweight clustering technique. Section 4.3.2 described a complex clustering and segmentation technique for combining a large surface into discrete components. When choosing clustering parameters a choice is made between erring on the side of over-segmentation (i.e., creating too many patches) or under-segmentation. Since it is generally easier to combine two discrete patches into one than it is to break an under-segmented component into two pieces, a default of over-segmentation is preferred. Unfortunately, due to the simplicity of the smart-merging technique compared to the more complex clustering and segmentation techniques, it often makes wrong decisions by merging together two patches that shouldn’t be merged. Thankfully, the technique allows for easy undo operations if the result is less than desirable.

Figure 68: Example of Hide Merging by (left) selecting a patch for consideration then (middle) hiding all patches that do not meet the distance, normal dot product, and coplanarity constraints. (right) The user can then manually choose which patches to merge with the patches still left displaying.

5.3.3. Hide Merging

Motivated by the techniques developed for smart merging (5.3.2), a more effective technique was created for assisting with the merging of many discrete surface patches. Hide merging essentially works the opposite of smart merging by simplifying the visual display through hiding all patches that are certainly not going to be merged with the patch under
consideration (Figure 68). The technique for determining which patches to hide is the same as in 5.3.2 only that the interpretation of the parameters are inverted. Figure 69 describes the relationship between smart merging and hide merging. In practice, hide merging is more useful than smart merging because it continues to provide users with a level of manual control that does not exist for smart merging. Since hide merging only limits the number of patches that are displayed, it can be thought of as a sort of occlusion rendering technique that limits the rendering of patches that are not relevant to the surface patch being investigated. The user is then still able to use domain knowledge about the nature of features being combined, albeit with less clutter on the screen.

Figure 69: Graph showing the relationship between smart merging and hide merging, making the assumption that an unknown optimal merge exists in the set of all surfaces. Smart merging parameters attempt to automatically merge less than the optimal, in order to prevent selecting too many patches. Hide merging parameters lie on the other side of optimal by trying to display more patches than optimal, in order to give the user a wider choice of patches to merge.

5.3.4. Separating Surfaces

The opposite of merging, separating features, is important such that features which were considered to be classified the same but are not can be separated from each other. Unfortunately, breaking apart features is a much more difficult technique to implement due to the fact that there are a much greater number of possible outcomes when breaking apart a feature as compared to only a limited number of ways a discrete number of features can be combined together. This large solution space comes from the problem with manually defining discriminations between parts of a complex 3-D feature. One way to accomplish this manual discrimination is to trace a line along
a feature, which forms a 2-D plane that can be used to separate one part of a feature from the rest, thereby breaking it into two pieces. Unfortunately, this approach assumes a very simple discrimination between features and makes it extremely difficult to define complex discriminations. In addition, in the case where hundreds of components need to be separated from a main feature this approach becomes very laborious and clumsy. Fortunately, techniques presented in section 4.3.2 are well suited to being integrated into an interactive technique for separating surface components.

![Select Patch](image1)

![Automatically Re-cluster](image2)

Figure 70: Example of surface breaking by selecting a green surface for processing (left) then breaking the surface into discrete patches with similar distance, normal dot product, and coplanarity arrangements as described by the different colors assigned to points (right).

The clustering technique described in section 4.3.2 can be adapted to an interactive approach where a selected surface feature is treated separate from all other visible features and it is broken into discrete pieces. As a user moves the mouse over an under-segmented surface feature, the breaking algorithm is initiated and the surface is re-clustered into smaller, discrete planar patches (see Figure 70). The junction points detected by the algorithm become the discriminations between different pieces of the surface and form the basis for the separations. The result is a set of discrete surfaces that are separated by junction points and combined when they share common planarity.

The combination of this surface breaking technique with the merging techniques described in sections 5.3.1, 5.3.2, and 5.3.3 creates a powerful system for interactively merging and breaking apart surface features in order to converge on a final solution. These techniques are complementary and work together such that if too much merging is conducted a surface can be
broken apart, or if a surface is broken into too many pieces it can be merged back together until a satisfactory solution is reached.

Figure 71: Evolution of fault based on a manual seed, followed by merging and surface creation.

5.4. Modifying Surfaces and Interactive Steering

Previous sections have described techniques for level set methods that model a wide variety of features, but there still exists a need to modify these evolutions in an interactive setting. The interactive guiding of level set evolution, also called interactive steering, can be accomplished through careful modifications of the velocity function. Applying user-defined control to the level set surface in this way allows growth and shrinkage in specific, which is the desired functionality.

Interactive steering is implemented using a shaped 3-D paintbrush, which defines the region of the surface where growing and shrinking occurs. Since both the implicit surface and the propagation function are stored in a volumetric format, there are two potential ways to approach
this topic. The first approach is to modify the surface directly by applying the 3-D paintbrush to the implicit surface volume. This requires dynamically modifying the distance transform representation of the implicit surface in order to redefine the zero-valued surface to encompass the changes made by the paintbrush. The implementation of this approach requires a reinitialization of the distance transform representation such that the user-defined modifications are treated as a zero crossing that is intersected with the implicit surface. In the case of growth the logical union of the zero crossings is the result and in the case of shrinkage the result is the logical difference of the implicit surface zero crossing with the zero crossing of the user-defined modification. The background motivation for this approach based on CSG modeling was described in section 2.4.8. After the combination of zero crossings is complete, the domain needs to be reinitialized so that it again correctly represents a distance transform volume.

Figure 72: Computational steering by interactively removing growth regions of the surface.

An alternative way to implement growth and shrinkage based on user-defined modifications is to work indirectly by changing the underlying velocity function, which in turn modifies the implicit surface (after a few iterations). This is the approach that was developed for this thesis due to its simplicity and speed. Since the first approach requires recomputing the distance transform, a computationally expensive move, it is usually preferable to compute the surface evolution for extra iterations in order to encompass the modifications via the velocity
function. Although, there is a point at which recomputing the distance transform volume requires less computation. This occurs when user-defined modifications significantly modify the implicit surface to the point that a large number of iterations would be necessary to evolve the surface into a new shape.

The velocity function modifying approach works by using the 3-D paintbrush to directly assign velocity values to the volume representing the evolution velocity. In the case of growth, positive propagation values are assigned by the paintbrush to the velocity volume, and in the case of shrinkage negative propagation values are used to retract the surface. This allows for the real-time modification of the surface as shown in Figure 73 for growing and Figure 72 for shrinking using an elongated cubic paintbrush. This technique finds use for preventing a surface from evolving into an incorrect region of the dataset or for encouraging the surface to evolve into a region of the dataset that it would not otherwise. Allowing all the interaction to take place in real-time fully represents a working implementation of computational steering.

Figure 73: Computational steering by interactively adding growth regions to the surface.

The techniques presented for level set modeling based on the paradigm where an initial seed computes a final solution surface needs to be expanded for situations where a surface requires further evolution in specific regions in order to fully describe a feature. Therefore, a
technique is presented for allowing an existing implicit surface to expand or shrink in specific regions while the remainder of the surface remains fixed. Instead of using a shaped 3-D paintbrush to accomplish this approach, the technique described for manual seed points using a point set dropper can be employed (section 5.2.3). Since there is already an existing representation of the implicit surface available, seed points can be drawn directly on the region of the implicit surface where growing or shrinking should occur (see Figure 74, left). After these seeds are defined a new implicit surface is created and evolved using one of the velocity functions presented in this thesis. Next, when the specific region finishes evolution, it is merged with the original implicit surface and a new distance transform is computed to generate the final surface (see Figure 74, right). This steering technique allows the interactive modification of surfaces by restricting evolution to user-defined parts of a surface in order to fully represent features.

Figure 74: Left to right, adding blue seed points to the edge of a surface then evolving it for 30 iterations. Result is an extended version of the implicit surface.

5.5. Usability and Validation

In order to investigate the usability and validation of the techniques presented in this thesis for interpreting 3-D seismic datasets, a number of studies were conducted. First, unconventional interaction techniques were explored such as speech-recognition and stereo visualization in order to investigate their value for the workflows presented in this work. The discussion of unconventional interaction techniques is pertinent since they can be readily applied to this work by replacing the traditional keyboard and mouse with speech recognition and replacing 2-D displays with a 3-D stereo display. Next, using the techniques developed for this
thesis, a seismic interpretation user-study was conducted by asking human participants to segment geologic surfaces. The experiments were conducted on the interpretation of a heavily faulted geologic region (section 5.5.4) and the extraction of geobodies (section 5.5.5). A final analysis is conducted on the value of interaction for seismic interpretation, in an attempt to quantify the importance of interactive “visulation”.

Figure 75: Example workflow of the interactive “visulation” system described in this thesis.

5.5.1. Unconventional Interaction Techniques

New user interaction devices such as speech recognition and 3-D stereovision can be replacements for the traditional keyboard/mouse and 2-D displays. Since interpreting seismic datasets is a very involved process requiring the attention of two hands and the user’s eyes, interpretation systems are limited in the amount of attainable interactivity. When interpreting datasets, users are often observed as voicing conversational commands [72] to the computer screen and rotating their heads in relation to the computer screen while interacting with data. This section will describe simple ways to exploit this human behavior through a paradigm that allows the computer to listen to vocal commands given by the user, such as “merge!” or “grow!” or “cluster!” Additionally, by using passive stereo technology available in new DLP displays, users should be able to achieve faster and more accurate problem solving for 3-D seismic interpretation.
through a more realistic 3-D environment. These possibilities are considered in the next two sub-sections.

5.5.1.1. **Speech-Guided Interpretation**

Speech recognition (also called voice recognition) technology converts human-spoken words to computer commands. Initial goals of speech recognition were to replace the need for keyboards within years of the technology’s inception. Although the day may never come when keyboards are obsolete, speech recognition is currently being widely used for applications of speech-input typing, such as generating transcriptions in the doctor’s office, and for hands-free operation of mobile phones [26]. Following this work, transcription could be very useful for geoscientists making notes when analyzing datasets, but this thesis is more interested in the use of speech to act as commands issued to an interpretation system. Hands-free mobile phones typically perform best at recognizing speech when providing a limited dictionary of commands to the user. Following this paradigm, a framework is proposed based on a limited number of commands that can be spoken to and interpreted by the system in the same way as a keystroke.

A significant amount of past work in speech recognition has been conducted for use in high-performance fighter aircraft. The US, French, and UK Militaries [37][38][140] all have research programs that have tried to determine where speech recognition would be helpful to pilots. Tasks such as setting radio frequencies, GPS coordinates, and steering directions, as well as deploying weapons and controlling displays have all been considered in these studies. Conclusions reached from this previous work are that a high level of recognition accuracy is far more important than a large vocabulary of commands, and therefore efforts should be focused on optimizing a limited vocabulary of speech recognition. Although the application of interpreting geologic features has significantly lower consequences in the case of a speech recognition system failing, as compared to a fighter pilot, applicable conclusions can still be gleaned from this work such as: speech recognition can be a powerful tool if it is natural to use and always works. In the
case where speech recognition has low reliability and is awkward to use, significant training will be required and users will become easily discouraged [63].

Putting these conclusions to work, five voice commands are defined to be recognized by software for the application of level set surface growth, which are listed in Table 4. These voice commands relate to interaction commands described previously in this thesis for enabling user interaction with an evolving surface. When working in voice command mode the computer’s recognition of these words relates to asserting the same keyboard command that would be used when voice command mode is disabled and the keyboard is the only input device. Voice commands are not attempted to replace the use of a mouse, just the keyboard.

<table>
<thead>
<tr>
<th>Word</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start!</td>
<td>start evolving surface</td>
</tr>
<tr>
<td>Stop!</td>
<td>stop evolving surface</td>
</tr>
<tr>
<td>Finish!</td>
<td>output final surface</td>
</tr>
<tr>
<td>Merge!</td>
<td>merge selected items</td>
</tr>
<tr>
<td>Break!</td>
<td>break-apart selected items</td>
</tr>
</tbody>
</table>

Table 4: List of voice commands accepted by software

The proof-of-concept implementation of the voice recognition software was implemented using Speakable Items [2] in the Apple OS 10.4 operating system and integrated with the CASI software package that has been used throughout this thesis. Apple Speakable Items allow spoken words to be processed by the computer and interpreted into commands. Therefore, a speech recognition system does not have to be implemented in order to test this concept since the operating system’s internal speech recognition is readily available. Since the CASI software used in this work is considered a 3rd-party application to OS 10.4, the application is limited to using Speakable Items to control keyboard commands. Therefore, five voice commands have been mapped to a set of distinct keyboard commands that will be recognized by CASI as a voice command. The voice commands used are shown in Table 4. It should be noted that Microsoft’s Windows Vista also has similar capabilities built into the operating system [88].
For the purposes of this thesis, the usability of speech-guided commands was determined by a subjective analysis of the proof-of-concept. Success can be simply determined if voice commands free the user’s hands from the keyboard so that he or she can focus on mouse interaction or other tasks. When segmenting a geologic feature using the techniques described in this thesis, instead of clicking a start button to begin the processing, the user can speak “start!” and the process begins. This allows the user to focus their attention on the visualization at the beginning of the segmentation, rather than awkwardly looking for the location of the start button in the GUI. In the same sense, when growth has reached a desired point of evolution, the user can command “stop!” to freeze the evolution. The process can proceed further by another “start!” and when a desirable result is achieved the command “finish!” stops the process and generates output. The user can then begin to combine the output into meaningful geologic features by moving the mouse cursor to select features and issuing a command to “merge!” when two should be combined into one, or “break!” when they should be separated into constituent components (section 5.3). Both of these grouping commands would otherwise require consulting the GUI and taking eyes off of the display, when voice control is not available.

Although a rigorous implementation and user study of speech recognition was not undertaken, some remarks on the topic can still be made. Using the existing speech recognition capabilities of operating systems is an easy way to integrate voice commands into a piece of software. The resulting interaction has significant promise as long as users are comfortable with speaking to their computer, something that may seem disturbing and unnatural for many people.

5.5.1.2. Stereo Vision for Seismic Interpretation

In this section, the possibility is discussed for using stereovision to aid in interpreting seismic features using techniques described in this thesis. The goal is not to provide an extensive analysis of 3-D stereo visualization, but instead to focus on a single method for easy visualization using passive stereo techniques.
In general, there are three types of devices used in practice for visualizing 3-D data, although there is an active research area focused on developing newer devices [10] [31]. Most often, 3-D data is displayed on a standard 2-D display, such as an LCD or CRT computer monitor. Three-dimensionality is achieved through the clever use of perspective views, shading, texture, and human imagination. There also exist devices capable of stereoscopic visualization, which works by sending two different views of 3-D data to the viewer’s two eyes. These stereoscopic displays can be generally divided into two types: passive or tracking. Stereoscopic displays using tracking work by enabling a head-tracking device to allow a person to “look around” 3-D data since moving their head will adjust the view automatically. Passive stereoscopic displays work by using glasses that allow different views of 3-D data to enter a person’s right eye and left eye. Passive stereo can be implemented using fairly simple equipment and software, and for that reason this section only focuses on passive stereovision. It should also be noted that 3-D visualization can be conducted in an immersive environment [132] such as a CAVE and it has been shown to provide added value for interpreting geologic data [46], although these immersive visualization systems are rare and not widely available.

In this work a Samsung DLP television is used that is capable of displaying stereo images synchronized to a person’s eyes using liquid crystal shutter glasses called CrystalEyes [107]. On the software side, the CASI visualization system described throughout this thesis continues to be used, which does all its visualization using OpenGL and is enabled to render stereo images (left and right eye). The goal for this section is to determine whether or not passive stereoscopic visualization can be an aid for using the 3-D techniques described previously in this thesis. Although it is beyond the scope of this thesis to exhaustively study this topic, the goal is to conduct a subjective analysis that could be used to encourage future work.

This work is focused on using the passive stereo display solely during a single part of the interpretation workflow that requires intensive 3-D interaction: the merging and separating of planar 3-D surfaces. This technology is introduced by first interpreting a fault dataset to the point
where the OpenGL render window is filled with colored surface components (Figure 65). Next, the stereo visualization modes in the software and the DLP television are enabled and the user puts on the CrystalEyes glasses. The user then merges and combines together sets of points by rotating the 3-D data while viewing in stereo. After about 5 minutes of operation the process is complete.

After having conducted this stereo merging multiple times on different datasets, then comparing the experience to working in non-stereo mode, it was lightly concluded that there was no significant change in interpretation speed using stereo as compared to non-stereo. Since this was only a proof of concept and a thorough study was not undertaken, significant conclusions should not be reached based on this limited experiment. It must be mentioned that although certain relationships in the 3-D orientations between sets of points did indeed appear more striking in stereo as compared to non-stereo; it did not result in increased efficiency. Although, further results gained from a full-fledged user study may determine that a decrease in interpretation time can result from using stereo visualization for the problem of merging sets of planar surfaces. This remains as a valid topic of future research.

5.5.2. User Studies Overview

The user studies conducted in sections 5.5.4 and 5.5.5, and the analysis of section 5.5.6 are designed to evaluate, in an applied setting, the techniques presented in this thesis. The expectation is that the studies will show how the combination of structure tensor analysis and level set segmentation with interactive 3-D visualization creates a tool that is faster, nearly as accurate, and more intuitive than manual segmentation techniques. In addition, results are expected that show a greater efficiency when using interactive techniques as compared to non-interactive techniques.

The purpose of these studies is to determine whether or not the proposed method can produce volumetric delineations comparable to features hand-picked by experts. Two geologic
features will be considered for segmentation: geobodies and faults. Due to the varied nature of these features in seismic data, further specification is needed. The geobodies will be interpreted from a seismic dataset that contains unique bodies based on seismic amplitude variations. Participants will be asked to interpret multiple geobodies in the same dataset. Faults will be interpreted from fault-likelihood volumes based on user-defined seed inputs. Participants will interpret multiple faults from the same dataset as well. Further details on the input datasets are given in following sections.

Due to the great expense required in acquiring data and the potential profits from knowing where oil is located, seismic datasets and the interpretations done on them are heavily protected pieces of information. In addition there are very few modern 3-D datasets freely available to the public. Any dataset that is available publicly will most certainly not have any interpretive information to go along with it, such as the locations of faults, channels, or geobodies. This poses a serious challenge to verifying techniques used to interpret geologic features in seismic datasets and greatly limits the number of data points that can be collected when verifying the accuracy of a new technique. Fortunately, TerraSpark Geosciences, a research partner with the University of Colorado Computer Science Department, has access to a limited number of 3-D seismic datasets that contain faults, channels, and geobodies. Unfortunately, unlike other industries like 3-D medical imaging it is rare to find a seismic dataset that has been fully interpreted multiple times. This does not necessarily mean that only one human conducted the interpretation, since it is often the case where a consensus among experts is reached before making an interpretation decision about a point in the data. What this does mean is that for any given seismic dataset, you will generally only find a single human interpretation. This fact holds true for the TerraSpark datasets and poses a problem when analyzing the results of techniques in this thesis. A final concern is that by doing a strict comparison between results the subjective nature of a human interpretation, which was generated with an unknown level of uncertainty, is
not being taken into account. Therefore a methodology is proposed for verifying interpretation results by taking into account uncertainties in the ground-truth.

### 5.5.3. Validation Methodology

Although the ideal approach to the verification of interpreted geologic features would be to compare results to the actual piece of Earth represented by the seismic data, there unfortunately exists a real-life problem with extracting physical data from the Earth’s crust. In the related field of 3-D medical imaging this is much less of a problem since it is possible to conduct studies on structures like kidneys inside of dead animals, and validity can be tested by physically extracting the kidney from the animal and measuring its volume via water displacement [15]. For living animals, less precise information is available although it is still possible to obtain precise surgical information on the size and shape of structures in a body. Now, there is sometimes real physical data available from the Earth in areas of seismic surveys containing previously drilled wells, which can be used to recognize rock types. Unfortunately these data points are extremely sparse (less than 0.0001% of the dataset) and are insufficient to reliably verify the identity of features inside the Earth. Therefore, it is not realistic to believe that this study can ever prove the correctness of geologic features being identified. For that reason, the goal of this study is to mimic the available expert human interpretation as closely as possible.

The method employed entails the consideration of three factors: validity, reliability, and efficiency. Validity [30] represents the accuracy to which the user results match the expert human interpretation. Reliability or reproducibility is a measure of precision that represents how closely user results match other user results. Efficiency is a measure of time that represents how long it takes for users to conduct the study as compared to the expert human interpretation.
Figure 76: Using the distance transform representation of different segmentations to create a combined segmentation that is also represented as a distance transform. The distance transform at level-zero represents the intersection of the segmentations. Using higher levels provides a consensus representation.

Previous work in validating volumetric segmentations has focused on techniques based on volume summation and the Hausdorff distance measure, although more complex methods have been proposed [133]. Techniques using volume summation [15] simply compute the sum of voxels contained in a template volume. This is the most simple technique and works well for the segmentation of features containing globular volumes where there is a high likelihood of intersection. This approach would be effective for certain channels and geobodies, but will often fail for planar features like faults that are likely to contain few regions of overlap in the case of a bad segmentation. The Hausdorff approach [58] measures how far two features are from each other based on minimizing a distance function between a template object and a sample. This technique is an improvement for comparing planar features, but the approach becomes more complicated when comparing objects of similar shape that are translations of each other such that there is no overlap of the shapes to be minimized.
Figure 77: Difference sample volume and template volume using the distance transform. The resulting zero crossing represents the discrimination between the two features, and higher levels of the combined distance transform create a consensus representation of the combined segmentations.

Fortunately, given the implicit surface representation of objects segmented in this thesis, a useful data representation is available that is exploited in a new technique for validating 3-D segmentations. Since objects in their implicit surface form are represented by their distance transform, this distance representation can be used to compare how far the zero-level sets of objects are from each other (see Figure 76). For expert interpreted features, which act as templates, they are not assumed to be in implicit surface form and therefore must have their distance transform computed in advance. In this approach, validity is quantified through the volumetric intersection of the distance transform of a sample object and the template. At the intersection of live-voxels, the distance value of the sample object is subtracted from the distance value of the template and a new combined value is stored into a volume. At the completion of this process, a new volume is created that represents the combination of two (or more) distance transforms such that the sum of the absolute value of all non-zero values represents the distance between the two shapes. Taking the intersection of an object with itself results in a value of zero, the desired result. As the sample interpretation moves further away from the template, the
difference in distance values becomes greater from more uncertainty, resulting in a greater measure of difference. This approach implicitly accounts for uncertainty in the expert interpretation since the distance transform represents a linear uncertainty scale that starts at zero and increases moving away from the template. An added benefit of this representation is that extracting the zero crossing of the difference volume creates surfaces that act as a discriminant between the two features (see Figure 77). This new technique overcomes the problems described for previous approaches and is a widely applicable for validating all types of volumetric segmentations.

Reproducibility or reliability is measured in a similar fashion, but this time comparisons are only made between the sample solutions created by participants (i.e., expert templates are not used). The method essentially follows the calculation for standard deviation using the volumetric validation technique described above. In order to establish a mean or average interpretation, the distance transforms of all sample features are combined into a single volume. This combined volume contains distance values represented such that a level set surface can be extracted at the locations of zero-crossings (see Figure 76). This new surface represents the combined mean surface of all sample volumes. After a mean interpretation is established, the standard deviation is calculated directly by computing the volumetric differences between all samples and the mean, using their distance transform representations. The resultant calculation can be represented in three-dimensions as a standard deviation surface (Figure 77), although for quantifying results it is easier to work with a singular value that represents the combined 3-D standard deviation. This single standard deviation value is the summed value of all voxel values in the standard deviation distance volume.
Figure 78: Combination between multiple volumetric interpretations (1, 2, ...n) using the distance transform.

Efficiency is measured by comparing the amount of time it takes participants to conduct the study as compared to the expert human interpretation. This is a situation where results need to consider the lack of data available from the expert human interpretations. Having only a single expert interpretation provides only one data point and does not take into account the obvious existence of variable speeds of interpreting data by experts at different levels of proficiency. This point needs to be considered when reviewing efficiency results in sections 5.5.4 and 5.5.5. The measure of efficiency is simply the average time taken for a user to complete the interpretation divided by the amount of time it takes an expert. Efficiency values of greater than 1.0 imply a speedup is achieved. The efficiency value can be co-analyzed with the accuracy values given by validity in order to create a correlated understanding showing changes in accuracy with respect to the amount of time a user takes to complete an interpretation. An additional point of analysis is
whether or not participants speed up their interpretations after gaining more experience with the system. This question is explored, in the context of interaction versus non-interaction, in section 5.5.6.

### 5.5.4. Study 1: Geologic Fault Segmentation

Study 1 was conducted on a seismic dataset taken out of the Gulf of Mexico that is given the name Volume G. The original survey size contains a voxel space of approximately 1000x1000x1000 at 8-bit resolution. The dataset contains hundreds of faults ranging in size from small faults a few voxels long to large faults that traverse most of the volume. This user study focuses on interpreting the extents of three of those larger faults, which have been previously interpreted by an expert using manual techniques. Volume G has been pre-processed using the smoothing technique in section 3.1 to remove noise and small discontinuities that do not describe faults. A fault likelihood algorithm then processed the volume such that the output contained bright values in the areas of faulting. Both the manual interpretation and the user study were conducted on this volume.

Eight participants were selected from University of Colorado students and staff to complete the study. None of the participants have previously used the software or are experts in seismic interpretation. The participants were briefly trained in the software and taught how to identify fault features from non-fault features using the preprocessed volume. The brief training included segmenting a fault from a dataset and adjusting parameters so their behavior could be understood. In order to guide the participants in the identification process, they were provided a single slice of the expert interpretation to use as a reference for identifying the locations of faults. The parameters exposed to the participants are listed in Table 5. Additional information on the user study is contained in Appendix D.1 such as the consent form and actual instructions given to each participant.
<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>100</td>
<td>Number of iterations surface evolves before completion</td>
</tr>
<tr>
<td>Propagation</td>
<td>1.0</td>
<td>Pushes the surface outwards and inwards according to feature</td>
</tr>
<tr>
<td>Curvature</td>
<td>1.0</td>
<td>Makes surface smooth by preventing leaks</td>
</tr>
<tr>
<td>Threshold</td>
<td>200</td>
<td>Voxel values greater will grow and lesser will shrink</td>
</tr>
</tbody>
</table>

Table 5: Parameters exposed to the participant during the fault segmentation user study

5.5.4.1. Fault Segmentation Workflow

The training given to participants described one of the fault segmentation workflows in the context of this thesis, in particular semi-automatic segmentation based on user-defined seeds. Techniques described in section 4.2 were implemented under-the-hood of the level set process. The participants were asked to only interpret the bounding surface of the fault (i.e., fault damage zone), which does not include medial surface extraction and segmentation. The study was conducted on a computer with two dual-core AMD 260s running Windows XP 64 with a single Nvidia 5600 graphics card. The individual faults are interpreted on a subvolume of the larger survey \((256^3)\), since the full dimensions of the survey are unwieldy. Half of the participants used an interactive version of the technique where surface evolution was visible at every timestep, while the other half of the participants used a non-interactive version where surface evolution was only visible at the final timestep. The participants were instructed to draw one or many seed points on one or many horizontal slices of the volume as a start to the segmentation. They were then instructed to observe the growth of the surface with respect to the data as it could be seen on slices through the volume. They could “pause” and “restart” the evolution at any time and changes to the parameters were handled in real-time. There was no time limit or restriction on the number of times a participant could restart the evolution, but participants were encouraged to work both quickly and accurately. When a satisfactory solution was attained, the output was saved before moving on to the next fault.
5.5.4.2. Results

Faults segmented by participants were saved to the computer disk and the amount of time taken to segment each fault was recorded on paper. Time was started when the participant was directed to the location of each fault and stopped when they were satisfied with the solution obtained. The investigator conducting the study sat alongside each participant in order to provide immediate answers to any questions or difficulties faced in working with the software. There were very few times when help was necessary and it primarily related to the participant forgetting how to do something trivial in the software that did not relate to the study.

Since section 5.5.6 investigates the value of interactivity in particular, the results in this section are combined between the participants using the interactive and non-interactive modes of the software. Table 6 shows the efficiency of this technique by comparing the amount of time it took participants to segment each fault as compared to the time taken by the manual interpretation. It should be obvious based on this chart that on average participants could segment features in a fraction of the time required to do so manually. Table 7 shows the validity of this technique by comparing how accurate the segmentations created by participants were compared
to the manual interpretation, assuming 100% accuracy for the manual interpretation. It can be seen that participants’ segmentations were generally accurate within 85-95% of the manual interpretation. Finally, Table 8 considers the deviation of participants’ segmentations from each other in order to determine the reliability of the technique. For each fault feature the deviation from the average of all participants’ segmentations was measured based on percent difference, and the results show an average similarity of 85%, 89%, and 96% for the three features. This low percent of average deviation suggests results are quite reproducible. Technical details on the calculation of the measures of validity and reproducibility were described in section 5.5.3.

Table 6: Time taken to interpret three fault features. Participants B, E, F used the non-interactive approach, while participants A, C, D, G, H used the interactive approach. Manual is the hand-drawn interpretations of the faults without using the techniques presented in this thesis.
Table 7: Accuracy of user-study results compared to the manual hand-drawn interpretation for the three fault features. Participants B, E, F used the non-interactive approach, while participants A, C, D, G, H used the interactive approach. A score of 100% accuracy is the same as identically matching the hand-drawn (manual) interpretations of the faults.

Table 8: Reliability of the technique as determined by the standard deviation of participants results. This is computed as the average percent differences between participant segmentations with the mean of all participant segmentations. A value of 0% means participant segmentations
were orthogonal to each other and a value of 100% means that all participants produced the exact same segmentation.

### 5.5.5. Study 2: Geobody Segmentation

Study 2 was conducted on the same seismic dataset from the Gulf of Mexico that is given the name Volume G. The original survey size contains a voxel space of around 1000x1000x1000 at 8-bit resolution. This user study focuses on interpreting the extents of three high-amplitude geobodies, which have been previously interpreted by an expert using manual techniques. Volume G has been pre-processed using the smoothing technique in section 3.1 along the seismic strata to remove noise and enhance layered features. Both the manual interpretation and the user study used the same preprocessed version of the seismic amplitude.

The same eight participants from University of Colorado Students and Staff completed this study immediately after Study 1. None of the participants had previously used the software or were experts in seismic interpretation. The participants were briefly trained in the software and taught how to identify high-amplitude geobodies from non-features using the preprocessed volume. The brief training included segmenting a geobody from a dataset and adjusting parameters so their behavior could be understood. In order to guide the participants in the identification process, they were provided a single slice of the expert interpretation to use as a reference for identifying the locations of geobodies. The parameters exposed to the participants are listed in Table 9.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>100</td>
<td>Number of iterations surface evolves before completion</td>
</tr>
<tr>
<td>Propagation</td>
<td>1.0</td>
<td>Pushes the surface outwards and inwards according to feature</td>
</tr>
<tr>
<td>Curvature</td>
<td>1.0</td>
<td>Makes surface smooth by preventing leaks</td>
</tr>
<tr>
<td>Threshold</td>
<td>127</td>
<td>Voxel values greater will grow and lesser will shrink</td>
</tr>
</tbody>
</table>

Table 9: Parameters exposed to the participant during the geobody segmentation user study
5.5.5.1. Geobody Workflow

The training given to participants described a geobody segmentation workflow in the context of this thesis. Simple geologic bodies (geobodies) were selected for this study that could be easily identified by a region of high amplitude in the seismic volume. The reason for choosing such a simple feature is to allow for a simplified workflow that can be learned by participants with minimal training. The study was conducted on the same workstation with two dual-core AMD 260s and running Windows XP 64 with a single Nvidia 5600 graphics card. The individual geobodies are interpreted from a $256^3$ subvolume of the larger survey. Half of the participants used an interactive version of the technique where surface evolution was visible at every time step, while the other half of the participants used a non-interactive version where surface evolution was only visible at the final time step. If participants used an interactive version during study 1, they also did so in this study. The participants were instructed to draw one or many seed points on one or many horizontal or vertical slices of the volume as a start to the segmentation. They were then instructed to observe the growth of the surface with respect to the data as it could be seen on slices through the volume. They could “pause” and “restart” the evolution at any time and changes to the parameters were handled in real-time. There was no time limit or limit on the number of times a participant could restart the evolution. When a satisfactory solution was attained, the output was saved before moving on to the next geobody.
Figure 80: (top to bottom) Expert segmentation of geobody surface (left) compared to multi-user combined result (right) for geobody features 1, 2, and 3 respectively.
5.5.5.2. **Results**

Analysis for this study closely resembles the analysis done in the previous study (section 5.5.4). Geobodies segmented by participants were saved to the computer disk as well as the amount of time taken to segment each geobody. Time was started when the participant was directed to the location of each feature and stopped when they were satisfied with the solution obtained. The investigator conducting the study sat alongside each participant in order to provide immediate answers to any questions or difficulties faced in working with the software. There were very few times when help was necessary and it primarily related to the participant forgetting how to do something in the software.

Since section 5.5.6 investigates the value of interactivity in particular, results in this section are combined between the participants using the interactive and non-interactive modes of the software. Table 10 shows the efficiency of this technique by comparing the amount of time it took participants to segment each geobody as compared to the time taken by the manual interpretation. It should be obvious based on this chart that on average participants could segment features in a fraction of the time required to do so manually. Table 11 shows the validity of this technique by comparing how accurate the segmentations created by participants were compared to the manual interpretation, assuming 100% accuracy for the manual interpretation. It can be seen that participants’ segmentations were generally accurate within 85-95% of the manual interpretation. Finally, Table 12 considers the deviation of participants’ segmentations from each other in order to determine the reliability of the technique. For each geobody feature the deviation from the average of all participants’ segmentations was measured based on percent difference, and the results show an average similarity of 98%, 97%, and 97% for the three features. This very low percent of average deviation (2%, 3%, 3%) suggests results are quite reproducible. Technical details on the calculation of the measures of validity and reproducibility were described in section 5.5.3.
Table 10: Time taken to interpret three geobody features. Participants B, E, F used the non-interactive approach while participants A, C, D, G, H used the interactive approach. Manual is the hand-drawn interpretations of the geobodies without using the techniques presented in this thesis.

Table 11: Accuracy of user-study results compared to the manual hand-drawn interpretation for the three geobody features. Participants B, E, F used the non-interactive approach while
participants A, C, D, G, H used the interactive approach. A score of 100% accuracy identically matches the hand-drawn (manual) interpretations of the geobodies.

Table 12: Reliability of the technique determined by the standard deviation of participants results. This is computed as the average percent differences between participant segmentations with the mean of all participant segmentations. A value of 0% means participant segmentations were orthogonal to each other and a value of 100% means that all participants resulted in the exact same segmentation.

5.5.6. Analyzing the Value of Interaction

An analysis was conducted in order to test the value of interaction for 3-D seismic interpretation. Interaction in this context is the combined action of computation and visualization where both have an effect on each other in a closed-loop system. The relationship is that computation creates the information that is visualized and the user’s response to the visualization adjusts the parameters of the computation. Interaction allows for an immediate response to data interrogations by visualization and should keep the user engaged, increase efficiency, and provide further understanding of the data. Through a deeper understanding of the data, users should be able to interpret additional features in less time when there are multiple features in the same
dataset. The value of this combined action for 3-D seismic interpretation is what will be analyzed in this section.

The value of interaction was analyzed during the user studies from sections 5.5.4 and 5.5.5. Participants in those studies were asked to segment a series of geologic features using either an interactive technique or a non-interactive technique. Two results are recorded: the amount of time required and the accuracy of the interpretation for participants in two groups (i.e., the interactive group and non-interactive group). The interactive technique enables isosurface visualization of the surface as it evolves, while the non-interactive technique only produces visualization when the computation is completed. Participants must determine when to stop the simulation before it goes too far and over-segments the data. The theory is that it should be much easier to determine the point at which the computations should stop when real-time visualization is available that allows rotating and zooming on the surface, as compared to waiting in batch-mode for a result to come back some time later after the user forgot what they were doing. The former describes the interactive group and the latter describes the non-interactive group.

Since earlier studies (5.5.4, 5.5.5) were conducted using both interactive and non-interactive techniques, there is sufficient data available to conduct this analysis without requiring additional studies. The information in tables Table 6, Table 7, Table 10, and Table 11 is used in order to create measures on the benefits of interactive visualization. The measures used compare for interactive and non-interactive techniques the time required to segment each feature (Table 13) followed by the accuracy of segmentations (Table 14).

Table 13 and 14 present the results of the user studies for fault and geobody segmentation by comparing interactive and non-interactive techniques. In Table 13 it can be seen that the average amount of time required to segment features (across all participants) using the interactive mode took significantly less time than required using the non-interactive mode, although it must be remembered that both modes were significantly faster than the manual technique (Table 6, Table 10). It can be concluded from this that being able to see the surface evolve interactively
allowed participants to reach conclusions faster as compared to blindly segmenting features where visualization is only available at the completion of the process (i.e., non-interactively). Table 13 plots the average segmentation time for each feature over the course of the study, from start to finish, in order to see if participants became more efficient as they gained more experience with the software. Looking at the slope of the lines it can be seen that the average times for participants working in interactive mode was mostly decreasing over the course of the study, suggesting that they became more efficient. By comparison, it is not obvious that participants working in non-interactive mode became more efficient to the same extent of the interactive mode. Unfortunately, we can not decisively make this conclusion since all participants segmented the same features in the same order, which does not account for any bias that may exist between the different features. Future work will therefore require conducting a similar study as done in section 5.5.4 or 5.5.5 where the order in which features are segmented is randomized. This will allow for a stronger conclusion to be made as to whether or not participants become more efficient over the course of a study when using interactive techniques.

Despite certain conclusions that cannot be reached, these results suggests that there is value to be gained from using interactive visualization; one reason for which may be that participants spent more time looking at evolving surfaces when working in interactive mode. By spending more time observing surfaces evolving in the data volume, participants may have gained a deeper understanding of the nature of the data, such as the unique shapes and characters of features. In addition, participants better understand how the implicit surface segmentation technique behaves when parameters are changed, since the effects of parameter changes are visible immediately. These same arguments can be applied to the results of Table 14, which show the accuracy of segmentations created using interactive and non-interactive techniques. In general, segmentations were more accurate using interactive techniques compared to non-interactive techniques, a result that can also be reasoned based on a better understanding of the data and technique parameters when using the interactive mode.
Table 13: Comparison of minutes required to complete segmentation for both interactive and non-interactive modes. It can be seen that participants working in interactive mode produced faster results on average.

Table 14: Differences in the validation between interactive and non-interactive techniques. It can be seen that segmentations computed using interactive techniques generally produced more accurate results.
6. Conclusions

6.1. Summary

In this work, dynamic implicit surfaces were implemented with level set methods in an Interactive “Visulation” Environment (IVE) to allow for the segmentation geologic surfaces from 3-D seismic datasets. The resulting techniques make significant contributions to the computational sciences and geologic sciences. In addition, the work presented here has all been implemented according to high standards of programming and are either being prepared for or are already available in a commercial seismic interpretation system. These contributions include structural smoothing to preserve stratigraphy in seismic datasets, a measure of curvature and confidence for identifying stratigraphic features, using tensor-derived attributes to steer a level set surface evolution, the computation of 2-D planar level sets in 3-D for fault modeling, the extraction and segmentation of the medial-surface of 3-D shapes, inverse curvature flow for shape enhancement, multiple interactive editing and steering of evolving implicit surfaces, and a streaming interactive GPU level set algorithm. The combination of these techniques is packaged into an ad-hoc system that presents a novel surface-driven approach to the interpretation of 3-D seismic data.

6.2. Future Work

During the course of conducting this research, a number of branches have been encountered along the way that were not fully pursued but are valid topics for future work. For instance, the fault segmentation technique described in this work uses the bounding surface of a fault plane during computation. This bounding surface was described as a natural representation for a popular component of faulting known as the fault damage zone (FDZ). The FDZ (or fault influence zone) is of interest to geoscientists since it is a description of the extent to which rock
layers are damaged near the fault. Although a fault is typically represented by a planar surface, there exists a number of applications that can benefit further from the region of influence caused by that fault. Since the technique presented in this work has that representation readily available, it can be immediately used in flow modeling applications that try to simulate how well a fluid will move through this zone. By integrating flow modeling through a FDZ with the segmentation technique presented in this work, a powerful tool would be available that takes geoscientists from seismic interpretation to hydrocarbon reservoir modeling in a unified approach. Although, research first needs to be conducted by comparing well logs with seismic data in order to verify that the bounding surface given by the level set representation of a fault is indeed describing the actual damage zone around a fault. If such a relationship can be established, this future work will be extremely valuable.

Most of the seismic datasets used in this work were acquired offshore in the Gulf of Mexico and therefore only represent a small region of the Earth’s geology. Although good results were achieved with these datasets, further work needs to be conducted in applying these techniques to several other 3-D datasets from different geologic regions. This is necessary to further establish the range of applicability of this work to a variety of geologic environments with varying noise and feature characteristics.

Prior research has been conducted in the area of parallel visualization and computation by using multiple GPUs on a single compute system [96] as well as parallel visualization on distributed GPU clusters [125]. It is foreseeable that the techniques described in this thesis could be mapped to either of these environments, which would theoretically allow for the interactive interpretation of datasets on a much larger scale than were considered here. A distributed visualization and computation environment would allow for the processing of entire seismic volumes on the scale of terabytes such that the use of subvolumes is unnecessary.

Another point of discussion with future work is applying the techniques presented here to fields other than seismic interpretation. As stated in the introduction, many techniques developed
in this work were motivated by prior work done in the 3-D medical imaging community and developed for the solution (ad-hoc) of interpreting geologic surfaces. It is foreseeable that medical imaging workflows would benefit from the structure analysis and complex surface deformations described in this work. In addition, simply using the IVE presented here for allowing the computational steering of surfaces towards features like vessels and tissue layers as well as tumors, organs, and ribbons would greatly benefit medical imaging workflows. Through the integration of surface modeling techniques with 3-D interaction methods, a similar package for the surface-driven analysis of 3-D medical images could be developed that make similar use of GPU technologies. Of particular interest in this area would be an expansion of the on-the-fly analysis techniques described in section 5.1.9 for dealing with real-time 3-D medical images where information must be continually processed and rendered immediately. This generalizes to dealing with 4-D data where the fourth dimension is time. Considering such datasets opens the door to a world of new research in the analysis of real-time 3-D data and time-evolved scientific simulations, such as climate modeling and mantle convection dynamics.
Bibliography

[60] Insight Segmentation and Registration Toolkit (ITK), ITK Software Guide 2.4.0, November, 2005.


International Conference 2008.

Geometry, Fluid Mechanics, Computer Vision, and Materials Science, Cambridge
vol. 93, nr. 4, pp 1591

segmentation and visualization of curvilinear structures in medical images, Lecture Note

application to thinning, IEE

Methods, and Real

Computer Society, Washington, DC, 179


Computer Society, Washington, DC, 179-188.


http://www.tatanano.com


[140] C.J. Weinstein, Opportunities for Advanced Speech Processing in Military Computer-Based Systems, Lincoln Laboratory, MIT, Lexington, MA, 19XX?.


Appendix A: Research Plan

A.1 Research Methodology

Research will be conducted at TerraSpark Geosciences LLP with advising from Henry Tufo. Programming will be done using Apple’s Xcode and Microsoft’s Visual Studio in the languages C, C++, OpenGL, and CUDA. Further details are outlines in the Appendices B.1, C.1, and D.1.

A.2 Research Schedule

<table>
<thead>
<tr>
<th>Entrance in Program</th>
<th>Proposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 2005</td>
<td>Undergraduate Thesis in Visualization of 3-D Mantle Convection Volumes</td>
</tr>
<tr>
<td>Summer 2005</td>
<td>Parallel Surface Extraction of NEK Simulations at NCAR</td>
</tr>
<tr>
<td>Aug. 2005</td>
<td>Begin Classes at CU</td>
</tr>
<tr>
<td>Fall 2005</td>
<td>Work at BP Center for Visualization in Modifying Automatic Fault Extraction</td>
</tr>
<tr>
<td>Spring 2006</td>
<td>Investigated 2-D Level Sets for Class Project in Computer Vision and HPC</td>
</tr>
<tr>
<td>Summer 2006</td>
<td>Used ITK Library to Investigate Level Sets in TerraSpark CASI Software</td>
</tr>
<tr>
<td>Fall 2006</td>
<td>Implemented Simple Fault Extraction Using Level Sets, Medial-Surface Ext.</td>
</tr>
<tr>
<td>Spring 2007</td>
<td>Narrow-Band Solver in CASI and Investigated Structure Tensors for Seismic</td>
</tr>
<tr>
<td>Summer 2007</td>
<td>Submitted WACV Confidence and Curvature Guided Level Sets</td>
</tr>
<tr>
<td>Fall 2007</td>
<td>Conduct work on Medial-Surfaces for Level Sets</td>
</tr>
<tr>
<td>Dec. 2007</td>
<td>Present Initial Research to Industry Consortia at 2007 GIVC Meeting</td>
</tr>
<tr>
<td>January 2008</td>
<td>Present Paper at WACV</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb/Mar 2008</td>
<td>Finished and Presented Proposal to PhD Committee</td>
</tr>
<tr>
<td>March 2008</td>
<td>Submit Papers on Medial-Surfaces and Shape Exaggeration to VIIP</td>
</tr>
<tr>
<td>April/May 08</td>
<td>Present at Poster at AAGP Conference, submit Level Set Exaggeration to VIIP</td>
</tr>
<tr>
<td>May 2008</td>
<td>Submit Paper to SEG on Channel Segmentation</td>
</tr>
<tr>
<td>May/June 08</td>
<td>Implement Streaming Level Set for G80 GPU</td>
</tr>
<tr>
<td>July 2008</td>
<td>Begin Work on Implicit Surface Merging/Editing Functionality</td>
</tr>
<tr>
<td>Sept 2008</td>
<td>Present Papers at VIIP</td>
</tr>
<tr>
<td>Oct. 2008</td>
<td>Design and Conduct User Studies</td>
</tr>
<tr>
<td>Nov. 2008</td>
<td>Present Paper at SEG</td>
</tr>
<tr>
<td>Nov/Dec 08</td>
<td>Finish Written Thesis</td>
</tr>
<tr>
<td>Dec. 2008</td>
<td>Present Research Results to Industry Consortia at 2008 GIVC Meeting</td>
</tr>
<tr>
<td>Dec 2008</td>
<td>Submit Final Thesis</td>
</tr>
<tr>
<td>Feb. 2009</td>
<td>Revise Thesis</td>
</tr>
<tr>
<td>March 2009</td>
<td>Defend Thesis in front of PhD Committee</td>
</tr>
<tr>
<td>March 2009</td>
<td>Submit Interactive Level Set/GPU Paper to IEEE Visualization</td>
</tr>
<tr>
<td>March 2009</td>
<td>Present Poster at 10th LCI Conference</td>
</tr>
<tr>
<td>April 2009</td>
<td>Submit Paper to Geophysics on Interactive Segmentation</td>
</tr>
</tbody>
</table>
Appendix B: Numerical Discretizations

This section contains code segments and further technical details for the various techniques described in this thesis. The code in this appendix derives from multiple sources and when taken from other sources, references are given.

B.1 Introduction

This appendix describes the numerical discretization of Equation 2-3, Equation 2-5, and Equation 2-13. The following code describes a paradigm used for calculating the locations of neighbors in a 3-D volume:

```c
// Numbering Scheme for neighborhood pixels along three slices
// 'p' designates z slice z+1
// 'm' designates z slice z-1
// -----------
// | 6 | 7 | 8 |
// | 3 | 4 | 5 |
// | 0 | 1 | 2 |
// -----------

float sampleValue = InputVol(i, j, k);

// Load values for Z slice
float u0, u1, u2, u3, u4, u5, u6, u7, u8;
u0 = InputVol(i-1, j-1, k);
u1 = InputVol(i, j-1, k);
u2 = InputVol(i+1, j-1, k);
u3 = InputVol(i-1, j, k);
u4 = InputVol(i, j, k);
u5 = InputVol(i+1, j, k);
u6 = InputVol(i-1, j+1, k);
u7 = InputVol(i, j+1, k);
u8 = InputVol(i+1, j+1, k);

// Load values for Z+1 slice
float u0_p, u1_p, u2_p, u3_p, u4_p, u5_p, u6_p, u7_p, u8_p;
u0_p = InputVol(i-1, j-1, k+1);
u1_p = InputVol(i, j-1, k+1);
u2_p = InputVol(i+1, j-1, k+1);
u3_p = InputVol(i-1, j, k+1);
u4_p = InputVol(i, j, k+1);
u5_p = InputVol(i+1, j, k+1);
u6_p = InputVol(i-1, j+1, k+1);
u7_p = InputVol(i, j+1, k+1);
u8_p = InputVol(i+1, j+1, k+1);

// Load values for Z-1 slice
float u0_m, u1_m, u2_m, u3_m, u4_m, u5_m, u6_m, u7_m, u8_m;
u0_m = InputVol(i-1, j-1, k-1);
u1_m = InputVol(i, j-1, k-1);
u2_m = InputVol(i+1, j-1, k-1);
u3_m = InputVol(i-1, j, k-1);
u4_m = InputVol(i, j, k-1);
u5_m = InputVol(i+1, j, k-1);
u6_m = InputVol(i-1, j+1, k-1);
u7_m = InputVol(i, j+1, k-1);
u8_m = InputVol(i+1, j+1, k-1);
```
B.2 Gaussian Convolution

This appendix describes the technique for computing the Gaussian convolution that is used for data filtering during finite differencing and the smoothing of structure tensor components. The Gaussian equation is given by the standard equation

\[
G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}
\]

and the convolution with an image \( I \): \( G \ast I \).

\[
G \ast I[x] = \sum_{-h}^{h} G[h]I[x + h]
\]

The convolution operator moves across the data along 1-D components for the entire 3-D volume. The Gaussian with \( \sigma \) value 2.0 can be visualized in 2-D as a surface or a matrix of values as shown below:

```
0.0001 0.0002 0.0006 0.0011 0.0015 0.0017 0.0015 0.0011 0.0006 0.0002 0.0001
0.0002 0.0007 0.0017 0.0033 0.0048 0.0054 0.0048 0.0033 0.0017 0.0007 0.0002
0.0006 0.0017 0.0042 0.0078 0.0114 0.0129 0.0114 0.0078 0.0042 0.0017 0.0006
0.0011 0.0033 0.0078 0.0146 0.0213 0.0241 0.0213 0.0146 0.0078 0.0033 0.0011
0.0015 0.0048 0.0114 0.0213 0.0310 0.0351 0.0310 0.0213 0.0114 0.0048 0.0015
0.0017 0.0054 0.0129 0.0241 0.0351 0.0398 0.0351 0.0241 0.0129 0.0054 0.0017
0.0015 0.0048 0.0114 0.0213 0.0310 0.0351 0.0310 0.0213 0.0114 0.0048 0.0015
0.0011 0.0033 0.0078 0.0146 0.0213 0.0241 0.0213 0.0146 0.0078 0.0033 0.0011
0.0006 0.0017 0.0042 0.0078 0.0114 0.0129 0.0114 0.0078 0.0042 0.0017 0.0006
0.0002 0.0007 0.0017 0.0033 0.0048 0.0054 0.0048 0.0033 0.0017 0.0007 0.0002
0.0001 0.0002 0.0006 0.0011 0.0015 0.0017 0.0015 0.0011 0.0006 0.0002 0.0001
```
The Gaussian operator is created using the following code segment where it is applied using convolution to an array of data:

```c
// Implements a 1-D convolution with step (h) and sigma (sigma).
double *kg = new double[(2*h)+1];
for(i=0; i<(2*h)+1; i++)
    kg[i] = (1/(2*M_PI*pow(sigmaD,2.0f)))*exp(-((i-h)^2)/(2*pow(sigma,2.0f)));

for(i=0; i<dataSize; i++)
    for(j=-h; j<=h; j++)
        OutputData[i] += Data[i+j] * kg[j];
```

### B.3 Gradient Structure Tensor

The 3-D gradient structure tensor for a volume $I$ is defined as follows:

$$G_{ST} = \begin{bmatrix} I_x^2 & I_xI_y & I_xI_z \\
I_xI_y & I_y^2 & I_yI_z \\
I_xI_z & I_yI_z & I_z^2 \end{bmatrix}$$

$$I_x = (I(x+1,y,z) - I(x-1,y,z)) \times 0.5$$

$$I_y = (I(x,y+1,z) - I(x,y-1,z)) \times 0.5$$

$$I_z = (I(x,y,z+1) - I(x,y,z-1)) \times 0.5$$

C++ code to compute the first order derivatives for the gradient structure tensor is described in the following segment:

```c
int computeGradientTensor(int i, int j, int k, volume *InputVol, tensor *Tensor)
{

    // Central Difference Derivatives
    Dx = (u5 - u3)*(0.5);
    Dy = (u7 - u1)*(0.5);
    Dz = (u4_p - u4_m)*(0.5);

    // Store into texture
    Tensor[0][0] = Dx*Dx;
    Tensor[1][1] = Dy*Dy;
    Tensor[2][2] = Dz*Dz;
    Tensor[0][1] = Dx*Dy;
    Tensor[0][2] = Dx*Dz;
    Tensor[1][2] = Dy*Dz;

    return 1;
}
```

### B.4 Hessian Tensor
The 3-D Hessian tensor for a volume $I$ is defined using second order derivatives as follows:

$$H_{ST} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

- $I_{xx} = I(x+1,y,z) - 2I(x,y,z) + I(x-1,y,z)$
- $I_{yy} = I(x,y+1,z) - 2I(x,y,z) + I(x,y-1,z)$
- $I_{zz} = I(x,y,z+1) - 2I(x,y,z) + I(x,y,z-1)$
- $I_{xy} = I(x+1,y,z) + I(x-1,y,z) + I(x,y+1,z) + I(x,y-1,z) - 4I(x,y,z)$
- $I_{xz} = I(x+1,y,z) + I(x-1,y,z) + I(x,y,z+1) + I(x,y,z-1) - 4I(x,y,z)$
- $I_{yz} = I(x,y+1,z) + I(x,y-1,z) + I(x,y,z+1) + I(x,y,z-1) - 4I(x,y,z)$

C++ code to compute the second order derivatives for the Hessian tensor is described in the following segment:

```cpp
int computeHessianTensor(int i, int j, int k, volume *InputVol, tensor *Tensor) {
    // Compute Partial Derivatives
    float Dxx, Dyy, Dzz, Dxy, Dxz, Dyz;

    // Derivatives
    Dxx = u5 + u3 - 2.0 * sampleValue;
    Dxy = 0.25 *( u0 - u6 - u2 + u8);
    Dxz = 0.25 *( u3_m - u3_p - u5_m + u5_p);
    Dyy = u7 + u1 - 2.0 * sampleValue;
    Dyz = 0.25 *( u1_m - u1_p - u7_m + u7_p);
    Dzz = u4_p + u4_m - 2.0 * sampleValue;

    // Store into texture
    Tensor[0][0] = Dxx;
    Tensor[1][1] = Dyy;
    Tensor[2][2] = Dzz;
    Tensor[0][1] = Dxy;
    Tensor[0][2] = Dxz;
    Tensor[1][2] = Dyz;

    return 1;
}
```

### B.5 Structure Tensor Eigenanalysis

The eigenvalues and eigenvectors to the gradient and Hessian structure tensors are solved using an analytical diagonalization technique specific to positive-definite and symmetric Cartesian tensors. The structure tensor $T$ has three principal invariants $\{I_1, I_2, I_3\}$ that can be defined as follows
\[
I_1 = \text{Trace}(T) = D_{xx} + D_{yy} + D_{zz} = \lambda_1 + \lambda_2 + \lambda_3
\]
\[
I_2 = \left(D_{xx}D_{yy} + D_{xx}D_{zz} + D_{yy}D_{zz}\right) - \left(D_{xy}^2 + D_{xz}^2 + D_{yz}^2\right) = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1
\]
\[
I_3 = \det(T) = D_{xx}D_{yy}D_{zz} + 2D_{xx}D_{xz}D_{yz} - \left(D_{xy}^2D_{zz} + D_{xz}^2D_{yz} + D_{yz}^2D_{xx}\right) = \lambda_1\lambda_2\lambda_3
\]

Following from the technique described in [50] the sorted eigenvalues \((\lambda_1 > \lambda_2 > \lambda_3)\) can be computed as

\[
\lambda_1 = I_1 / 3 + 2\sqrt{b} \cos(\phi)
\]
\[
\lambda_2 = I_1 / 3 - 2\sqrt{b} \cos(\pi/3 + \phi)
\]
\[
\lambda_3 = I_1 / 3 - 2\sqrt{b} \cos(\pi/3 - \phi)
\]

After the sorted eigenvalues have been computed, the calculation of the corresponding eigenvectors proceeds as follows, simplifying for the \(i\)th components

\[
A_i = D_{xx} - \lambda_i
\]
\[
B_i = D_{yy} - \lambda_i
\]
\[
C_i = D_{zz} - \lambda_i
\]

the \(i\)th eigenvector is computed as

\[
e_{ix} = (D_{xy}D_{yz} - B_iD_{xz})(D_{xz}D_{yz} - C_iD_{xy})
\]
\[
e_{iy} = (D_{xz}D_{yz} - C_iD_{xy})(D_{xz}D_{xy} - A_iD_{yz})
\]
\[
e_{iz} = (D_{xy}D_{yz} - B_iD_{xz})(D_{xz}D_{xy} - A_iD_{yz})
\]

and in normalized form

\[
\hat{e} = \frac{e}{|e|}
\]

An alternative approach to finding the third eigenvector would be to compute the cross product of the first two eigenvectors.

The following C++ code segment, taken from Jeong et al. [61], computes the eigenvectors and eigenvalues of a positive-definite tensor (such as the GST or HST).

```c++
{
    // Loading tensor (gradient or hessian) components
    float xx = Tensor[0][0];
    float xy = Tensor[0][1];
    float xz = Tensor[0][2];
    float yy = Tensor[1][1];
    float yz = Tensor[1][2];
    float zz = Tensor[2][2];

    float b = -(xx + yy + zz);

    float c = xx * yy + yy * zz +
              xx * zz - xy * xy -
```
yz * yz - xy * xy;

float d = xy * xy * zz +
yz * yz * xx +
xy * xy * yy -
xx * yy * zz -
xy * yz * xy -
xy * xy * yz;

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;

float evalx, evaly, evalz;
if(f == 0.0 && g == 0.0 && h == 0.0) {
  evalx = evaly = evalz = -pow(d, float(1.0/3.0));
}
else {
  float i = sqrt((g*g)/4.0 - h);
  float j = pow(i, float(1.0/3));
  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);
  evalx = 2.0*j*cos(k/3.0) - (b/3.0);
  evaly = -j*(m-n)+p;
  evalz = -j*(m+n)+p;
}

float A[3], B[3], C[3], D[3], E[3], F[3];
A[0] = xx - evalx;
B[0] = yy - evalx;
B[1] = yy - evaly;
C[0] = zz - evalx;
C[1] = zz - evaly;
C[2] = zz - evalz;
D[0] = xy*yz - B[0]*xy;
D[1] = xy*yz - B[1]*xy;
E[0] = xy*yz - C[0]*xy;
E[1] = xy*yz - C[1]*xy;
F[0] = xy*xy - A[0]*yz;

// Eigenvectors
float evec0[3], evec1[3], evec2[3];
B.6 Smoothing Tensor Components

The smoothing of tensor components using Gaussian convolution provides a more robust orientation field. Following from appendix A.1 where Gaussian smoothed finite differences were used to compute the structure tensor, the component $i$ of the smoothed structure tensor is defined as

$$\bar{T}_i = G \ast T_i = \sum_{k=-\text{var}}^{\text{var}} G[k] T_i[x + k]$$

In a similar fashion as in A.1, the smoothed tensor components can be calculated using the following code segment.

```cpp
// Implements a 1-D convolution with step (h) and sigma (sigma).
double *kg = new double[(2*h)+1];
for(i=0; i<2*h+1; i++){
    kg[i] = (1/(2*M_PI*pow(sigmaD,2.0f)))*exp(-((i-h)^2)/(2*pow(sigma,2.0f)));
}
for(i=0; i<dataSize; i++){  
    for(j=-h; j<=h; j++){  
        SmoothTensor[i][0][0] += kg[j] * Tensor[i+j][0][0];  
        SmoothTensor[i][1][1] += kg[j] * Tensor[i+j][1][1];  
        SmoothTensor[i][2][2] += kg[j] * Tensor[i+j][2][2];  
        SmoothTensor[i][0][1] += kg[j] * Tensor[i+j][0][1];  
        SmoothTensor[i][0][2] += kg[j] * Tensor[i+j][0][2];  
        SmoothTensor[i][1][2] += kg[j] * Tensor[i+j][1][2];  
    }
}
```

B.7 Level Set Methods

The level set update scheme requires the computation of a number of derivatives including first and second order derivatives described in A.2 and A.3. The following discretization is taken from Lefohn [82].

The two normals at the point $x$, $y$, $z$ on the level set image $I$ are defined using the upwind derivatives computed as
The gradient of the level set \( \phi \) is computed using the upwind derivatives for growth and shrinkage as

which are used for the computation of the normals (+/-) as

\[
N_x^+ = \frac{I_x^+}{\sqrt{(I_x^+)^2 + \left(\frac{I_x^++I_y}{2}\right)^2 + \left(\frac{I_x^++I_z}{2}\right)^2}} \\
N_y^+ = \frac{I_y^+}{\sqrt{(I_y^+)^2 + \left(\frac{I_x^++I_y}{2}\right)^2 + \left(\frac{I_x^++I_z}{2}\right)^2}} \\
N_z^+ = \frac{I_z^+}{\sqrt{(I_z^+)^2 + \left(\frac{I_x^++I_y}{2}\right)^2 + \left(\frac{I_x^++I_z}{2}\right)^2}}
\]

\[
N_x^- = \frac{I_x^-}{\sqrt{(I_x^-)^2 + \left(\frac{I_x^-+I_y}{2}\right)^2 + \left(\frac{I_x^-+I_z}{2}\right)^2}} \\
N_y^- = \frac{I_y^-}{\sqrt{(I_y^-)^2 + \left(\frac{I_x^-+I_y}{2}\right)^2 + \left(\frac{I_x^-+I_z}{2}\right)^2}} \\
N_z^- = \frac{I_z^-}{\sqrt{(I_z^-)^2 + \left(\frac{I_x^-+I_y}{2}\right)^2 + \left(\frac{I_x^-+I_z}{2}\right)^2}}
\]

The mean curvature is then computed as the divergence of the unit normal

\[
\kappa = 0.5 \left( (N_x^+ - N_x^-) + (N_y^+ - N_y^-) + (N_z^+ - N_z^-) \right)
\]

The gradient of the level set \( \phi \) is computed using the upwind derivatives for growth and shrinkage as
\[ \nabla \phi^\text{grow}_x = \sqrt{\max(I_x^+, 0)^2 + \max(-I_x^-, 0)^2} \]
\[ \nabla \phi^\text{shrink}_x = \sqrt{\min(I_x^+, 0)^2 + \min(-I_x^-, 0)^2} \]
\[ \nabla \phi^\text{grow}_y = \sqrt{\max(I_y^+, 0)^2 + \max(-I_y^-, 0)^2} \]
\[ \nabla \phi^\text{shrink}_y = \sqrt{\min(I_y^+, 0)^2 + \min(-I_y^-, 0)^2} \]
\[ \nabla \phi^\text{grow}_z = \sqrt{\max(I_z^+, 0)^2 + \max(-I_z^-, 0)^2} \]
\[ \nabla \phi^\text{shrink}_z = \sqrt{\min(I_z^+, 0)^2 + \min(-I_z^-, 0)^2} \]

where the choice of gradient is based on the sum of all speed terms \( S \) such that

\[ \nabla \phi = \begin{cases} 
\nabla \phi^\text{grow} & S > 0 \\
\nabla \phi^\text{shrink} & S \leq 0
\end{cases} \]

The time step is calculated based on CFL conditions described in Section 2.4.3 as

\[ \Delta t = \frac{1}{6\left(\sum_i \max(S_i)\right)} \]

The following code segment, adapted from the Insight Toolkit [itk.org], is an implementation of the techniques for computing the level set update:

```cpp
float computeLevelSetVoxel(int i, int j, int k, volume *LSVol, volume *SpeedVol, volume *AdvectionVol) { 
    // Numbering Scheme for neighborhood pixels along three slices
    // 'p' designates z slice z+1
    // 'm' designates z slice z-1
    // -------------
    // | 6 | 7 | 8 |
    // | 3 | 4 | 5 |
    // | 0 | 1 | 2 |
    // -------------

    float sampleLevelSet = LSVol(i, j, k);
    float sampleSpeed = SpeedVol(i, j, k);

    // Load values for Z slice
    float u0, u1, u2, u3, u4, u5, u6, u7, u8; 
    u0 = LSVol(i-1, j-1, k); 
    u1 = LSVol(i, j-1, k); 
    u2 = LSVol(i+1, j-1, k); 
    u3 = LSVol(i-1, j, k); 
    u4 = LSVol(i, j, k); 
    u5 = LSVol(i+1, j, k); 
    u6 = LSVol(i-1, j+1, k); 
    u7 = LSVol(i, j+1, k); 
    u8 = LSVol(i+1, j+1, k);

    // Load values for Z+1 slice
    float u0_p, u1_p, u2_p, u3_p, u4_p, u5_p, u6_p, u7_p, u8_p; 
    u0_p = LSVol(i-1, j-1, k+1); 
    u1_p = LSVol(i, j-1, k+1); 
    u2_p = LSVol(i+1, j-1, k+1); 
    u3_p = LSVol(i-1, j, k+1); 
    u4_p = LSVol(i, j, k+1); 
    u5_p = LSVol(i+1, j, k+1); 
    u6_p = LSVol(i-1, j+1, k+1); 
    u7_p = LSVol(i, j+1, k+1); 
    u8_p = LSVol(i+1, j+1, k+1);
```
u2_p = LSVol(i+1, j-1, k+1);
u3_p = LSVol(i-1, j, k+1);
u4_p = LSVol(i, j, k+1);
u5_p = LSVol(i+1, j, k+1);
u6_p = LSVol(i-1, j+1, k+1);
u7_p = LSVol(i, j+1, k+1);
u8_p = LSVol(i+1, j+1, k+1);

// Load values for Z-1 slice
float u0_m, u1_m, u2_m, u3_m, u4_m, u5_m, u6_m, u7_m, u8_m;
u0_m = LSVol(i-1, j-1, k-1);
u1_m = LSVol(i, j-1, k-1);
u2_m = LSVol(i+1, j-1, k-1);
u3_m = LSVol(i-1, j, k-1);
u4_m = LSVol(i, j, k-1);
u5_m = LSVol(i+1, j, k-1);
u6_m = LSVol(i-1, j+1, k-1);
u7_m = LSVol(i, j+1, k-1);
u8_m = LSVol(i+1, j+1, k-1);

// Compute Every 3D Partial Derivative, offset is fixed at 1
float Dx, Dy, Dz;
float Dxx, Dyy, Dzz, Dxy, Dxz, Dyz;
float Dx_p, Dy_p, Dz_p, Dx_m, Dy_m, Dz_m;
float Dx_py, Dx_my, Dx_pz, Dx_mz;
float Dy_px, Dy_mx, Dy_pz, Dy_mz;
float Dz_px, Dz_mx, Dz_py, Dz_my;

// Central Difference Derivatives
Dx = (u5 - u3)*0.5;
Dy = (u7 - u1)*0.5;
Dz = (u4_p - u4_m)*0.5;

// Second-order Derivatives
Dxx = u5 + u3 - 2.0 * sampleLevelSet;
Dxy = 0.25 *(u0 - u6 - u2 + u8);
Dxz = 0.25 *(u3_m - u3_p - u5_m + u5_p);
Dyy = u7 + u1 - 2.0 * sampleLevelSet;
Dyz = 0.25 *(u1_m - u1_p - u7_m + u7_p);
Dzz = u4_p + u4_m - 2.0 * sampleLevelSet;

// Forward Difference Derivatives
Dx_p = u5 - u4;
Dy_p = u7 - u4;
Dz_p = u4_p - u4;

// Backward Difference Derivatives
Dx_m = u4 - u3;
Dy_m = u4 - u1;
Dz_m = u4 - u4_m;

// X w.r.t. Y-Forward/Backward Partial Derivatives
Dx_py = (u8 - u6)*0.5;
Dx_my = (u2 - u0)*0.5;

// X w.r.t. Z-Forward/Backward Partial Derivatives
Dx_pz = (u5_p - u3_p)*0.5;
Dx_mz = (u5_m - u3_m)*0.5;
// Y w.r.t. X - Forward/Backward Partial Derivatives
Dy_px = (u8 - u2)*(0.5);
Dy_mx = (u6 - u0)*(0.5);

// Y w.r.t. Z - Forward/Backward Partial Derivatives
Dy_pz = (u7_p - u1_p)*(0.5);
Dy_mz = (u7_m - u1_m)*(0.5);

// Z w.r.t. X - Forward/Backward Partial Derivatives
Dz_px = (u5_p - u5_m)*(0.5);
Dz_mx = (u3_p - u3_m)*(0.5);

// Z w.r.t. Y - Forward/Backward Partial Derivatives
Dz_py = (u7_p - u7_m)*(0.5);
Dz_my = (u1_p - u1_m)*(0.5);

////////////////////////////////
// Compute Second Derivatives
float n_p[3], n_m[3];
float Dn_x, Dn_y, Dn_z;
n_p[0] = (Dx_p / sqrt(Dx_p*Dx_p + ((Dy_px + Dy)*0.5)*((Dy_px + Dy)*0.5) + ((Dz_px + Dz)*0.5)*((Dz_px + Dz)*0.5)));
n_p[1] = (Dy_p / sqrt(Dy_p*Dy_p + ((Dx_py + Dx)*0.5)*((Dx_py + Dx)*0.5) + ((Dz_py + Dz)*0.5)*((Dz_py + Dz)*0.5)));

n_p[2] = (Dz_p / sqrt(Dz_p*Dz_p + ((Dx_pz + Dx)*0.5)*((Dx_pz + Dx)*0.5) + ((Dy_pz + Dy)*0.5)*((Dy_pz + Dy)*0.5)));
n_m[0] = (Dx_m / sqrt(Dx_m*Dx_m + ((Dy_mx + Dy)*0.5)*((Dy_mx + Dy)*0.5) + ((Dz_mx + Dz)*0.5)*((Dz_mx + Dz)*0.5)));
n_m[1] = (Dy_m / sqrt(Dy_m*Dy_m + ((Dx_my + Dx)*0.5)*((Dx_my + Dx)*0.5) + ((Dz_my + Dz)*0.5)*((Dz_my + Dz)*0.5)));

n_m[2] = (Dz_m / sqrt(Dz_m*Dz_m + ((Dx_mz + Dx)*0.5)*((Dx_mz + Dx)*0.5) + ((Dy_mz + Dy)*0.5)*((Dy_mz + Dy)*0.5)));

Dn_x = n_p[0] - n_m[0];
Dn_y = n_p[1] - n_m[1];
Dn_z = n_p[2] - n_m[2];

// Compute Mean Curvature of Level Sets
float H;
H = (0.5)*(Dn_x + Dn_y + Dn_z);
H = H/(Dx*Dx+Dy*Dy+Dz*Dz);

///////////////////////////////////////////////////////////////////////
// Computing the Level Set
// Upwind approximation to gradLS (gradient of Level Set), given by a Growth(max) and Shrinkage(min)
float gradLS_max[3], gradLS_min[3];
gradLS_max[0] = sqrt((float) max(Dx_p, 0.0)*max(Dx_p, 0.0) + max(-Dx_m, 0.0)*max(-Dx_m, 0.0));
gradLS_max[1] = sqrt((float) max(Dy_p, 0.0)*max(Dy_p, 0.0) + max(-Dy_m, 0.0)*max(-Dy_m, 0.0));
gradLS_max[2] = sqrt((float) max(Dz_p, 0.0)*max(Dz_p, 0.0) + max(-Dz_m, 0.0)*max(-Dz_m, 0.0));

gradLS_min[0] = sqrt((float) min(Dx_p, 0.0)*min(Dx_p, 0.0) + min(-Dx_m, 0.0)*min(-Dx_m, 0.0));
gradLS_min[1] = sqrt((float) min(Dy_p, 0.0)*min(Dy_p, 0.0) + min(-Dy_m, 0.0)*min(-Dy_m, 0.0));
gradLS_min[2] = sqrt((float) min(Dz_p, 0.0)*min(Dz_p, 0.0) + min(-Dz_m, 0.0)*min(-Dz_m, 0.0));
\[
\text{gradLS}_{\text{min}[2]} = \sqrt((\text{float} \min(Dz_p, 0.0)*\min(Dz_p, 0.0) + \min(-Dz_m, 0.0)*\min(-Dz_m, 0.0)));
\]

float S = sampleSpeed;
float A = advectionTerm;

// Calculate gradLS as the L2 norm of the appropriate gradLS vector
float gradLS;
if(S+A+H < 0.0)
    gradLS = sqrt( gradLS_{\text{max}[0]}*gradLS_{\text{max}[0]} + gradLS_{\text{max}[1]}*gradLS_{\text{max}[1]}
                   + gradLS_{\text{max}[2]}*gradLS_{\text{max}[2]} );
else
    gradLS = sqrt( gradLS_{\text{min}[0]}*gradLS_{\text{min}[0]} + gradLS_{\text{min}[1]}*gradLS_{\text{min}[1]}
                   + gradLS_{\text{min}[2]}*gradLS_{\text{min}[2] } );

// Compute dLSdt as combination of linear terms
float dLSdt = -gradLS*(propagationWeight*S) - (advectionWeight*A) + (curvatureWeight*H);

// Compute LS(t+dt) using upwind scheme
float LS_new;
LS_new = sampleLevelSet + dt*dLSdt;

// Update volume with new value
return LS_new;

}
Appendix C: Software Design

C.1 Introduction
This appendix describes the software design used in the implementation of the techniques presented in this thesis. Since this software was integrated into the commercial research product CASI from TerraSpark Geosciences, certain design details that are not relevant to this thesis work are omitted.

C.2 Design Overview
The graphical user interface and windowing library used as a platform for implementing the techniques in this work is the WxWidgets library (wxwidgets.org). Software engineers at TerraSpark Geosciences chose WxWidgets for use in the CASI software package, which made it a convenient choice in this work. WxWidgets is a free open-source software designed for use in cross-platform applications so that it can run in Windows, Mac OS X, and Linux. Being able to easily switch between different operating systems was advantageous for the work in this thesis due to the wide range of machines used to develop software and generate results in different labs. As a result, the software coming out of this thesis can be compiled to run on nearly any platform.

The C++ programming language was used to write the software in this work due to its compatibility with the windowing library, its object-orientated environment (compared to C), its high-performance (compared to Java), and its compatibility with GPU programming. Graphics were implemented using the OpenGL library (opengl.org), which is considered the industry standard API for cross-platform 3D graphics. Version 2.0 functionality of the OpenGL API is used in many places, which requires a relatively modern graphics card that is able to implement these new functions. The alternative to using OpenGL is DirectX, which is a similar API designed specifically for Microsoft platforms. Since this work is striving to be cross-platform compatible, DirectX was not considered.

In order to further the understanding of techniques developed in this thesis, every numeric solver and function was written from scratch. An alternative approach would have been to use a numerical library such as PETSc (wwunix.mcs.anl.gov/petsc), Netlib (netlib.org), or Insight Toolkit (itk.org), but that would have required following strict constraints in the development of algorithms to fit within predefined classes and result in creating too much abstraction from the nuts and bolts of algorithms. The from-scratch approach no doubt required significantly more development time but it can be easily argued that it allowed for a deeper understanding of the topics in this thesis.

On the visualization side, all the rendering routines were also written from scratch. Again, there was an alternative option of going with a library like the Visualization Toolkit (vtk.org) or OpenInventor (mercury.com). For the same reason as with the numerical libraries, the decision was made to work from scratch in order to enable easier functionality to be included in the software as well as being able to further understand the visualization techniques used, by having to fully implement them from the ground up.

For GPU programming, the opposite decision was made when going with CUDA as the API for writing GPU code. The bare-bones alternative approach to CUDA is writing vertex and shader programs in OpenGL to accomplish the same result as a program in CUDA. Although CUDA provides no pre-defined functions or library routines, it does allow for easier programming on a graphics card since it allows the developer to think in terms of standard C/C++ parlance instead of the graphics paradigm. It is expected that CUDA functionality will be included in the soon to be released OpenCL API, to allow for a more general implementation of techniques.
C.3 Data Types

Input data is assumed to be represented in a volumetric format commonly called a “brick of bytes”, which literally describes the arrangement of bytes in the data. In the commercial software world this is called a “dot vol” which is a reference to the common filename extension of “.vol” assigned to files of this type. This volume can be stored in either an 8-bit unsigned character format or a 32-bit floating point format. Volumetric data can be visualized using three different types of slices that cut through the volume along the three Cartesian coordinates and they are colored according to a color mapped intensity scale that matches values to colors.

Surfaces are represented as two tandem arrays representing the x,y,z coordinates of triangle points and the x,y,z components of each triangle’s normal vector. Connectivity information between triangles is not needed in this work and therefore it is not stored. Triangles can be visualized as wireframes or polygon surfaces.

C.4 Filter Layer

Data is processed using functions called filters or processes, both names are considered interchangeable for this purpose. Filters derive from a base class defined in CASI called the Filter class. The Filter class takes an input, does something to it, then create an output with the updated data. The VolumeFilter class derives from the Filter class and is intended for processing only on volumetric data. The level set segmentation techniques described in this work are members of the VolumeFilter class since they take as input a seismic volume and create a level set volume as output. The isosurface extraction techniques described in this thesis are members of a SurfaceFilter class that outputs a surface data type.

C.5 Numerics Layer

Numerical techniques are solely contained within the Filter class and are not represented in their own class. Although this may not be the best object oriented approach to take advantage of code reuse, by keeping numerics contained within each member of the Filter class it allows for easy transfer of techniques to other applications since class dependencies are limited. Therefore, multiple implementations of finite differencing, Gaussian convolution, distance transforms, zero crossing extractions, and isosurface extractions exist throughout the software, although their results are the same.

C.6 CUDA Layer

CUDA is an extension to the C programming language, but since the interface is not C++ it is not straightforward to integrate CUDA with the rest of this work. CUDA programs are stored in “.cu” files that need to be compiled using a special NVIDIA compiler. The compiled objects can then be linked into the main software and the functions accessed as external calls. This destroys much of the object oriented design and requires a situation where the CUDA layer stands on its own and cannot use functions defined in any of the CASI base classes.

C.7 Rendering Layer

The rendering of volumetric data is accomplished by creating OpenGL textures that represent the orthogonal slices through the volume and writing data from the volume into the texture for display. The rendering of surface data is accomplished by directly rendering the list of vertices as triangles where a new triangle begins every three points and the corresponding normal vectors are assigned to the triangles. The triangles can be rendered as polygonal surfaces or as wireframe surfaces.
Figure 81: The software design process combines into a segmentation application that consists of a data window, process window (workflows and algorithms), parameter window, and 3-D render window.

C.8 Segmentation Application

The above layers are combined into a segmentation application that implements techniques and visualization. Figure 81 shows a screenshot from the segmentation application that is used to present the techniques developed in this work to users. The application is divided into windows that manage the data, arrange processes icons to implement workflows, adjust parameters for process techniques, and visualize the data in 3-D.
Appendix D: Human Studies

D.1 Introduction
This section contains documentation and forms used for the human studies portion of this dissertation. This includes the user studies conducted in sections 5.5.4, 5.5.5, 5.5.6. The human studies conducted for this dissertation were approved on (11/29/2009) by the University of Colorado Human Studies review board using the request for review form in Figure 82. The user study was concluded not to be research on humans.

Figure 82: University of Colorado Human Studies Request for Review Form

D.2 Consent Form
The following consent form was signed by all participants in the user study (Figure 83):

Figure 83: Consent Form
D.3 Human Research Committee Determination Letter

Figure 84: University of Colorado Human Research Committee Determination Letter

D.4 Participant Instructions

The following outline of introduction and training were given to participants of the user study. It took about 5 minutes to present this information to each participant prior to his or her taking part in the study.
1. Description of 3-D seismic volume  
2. Description of 3-D fault volume  
3. How to pick values using mouse  
4. How to rotate and zoom the data  
5. How to select, hide, adjust slices  
6. How to place seed points  
7. Computing with default parameters  
8. Adjusting high/low thresholds  
9. Scaling propagation and curvature  
10. Computing with additional iterations  
11. Pausing, stopping, starting, restarting

Table 15: Summary of instructions given to participants being introduced to the user study.
Curriculum Vitae: Benjamin J. Kadlec (Feb. 2009)

PhD Candidate – University of Colorado (Expected Graduation: March 2009)
- Thesis: “Interactive GPU-Based ‘Visulation’ and Structure Analysis of Implicit Surfaces for Seismic Interpretation”

Professional Experience:
- Senior Research Software Developer – TerraSpark Geosciences (August 2005 - Present)
  - Developed a 3D Computer Aided Seismic Interpretation System (CASI) in C++ which performs data management, numerous patented algorithms and OpenGL 2.0 visualization. Along with three other team members who wrote the system, shipped the product to a number of major oil companies and service providers and support it commercially.
  - Conducted cross-platform development using wxWidgets GUI toolkit for commercial software release of CASI in Windows XP, Red Hat Linux, and Mac OS X environments.
  - Lead research developer for Automatic Fault Extraction, Channel Segmentation, and other proprietary algorithms, which have been submitted for patents and published in literature. These pieces of software have been implemented in CASI and licensed by a major oil service provider.
  - Implemented algorithms in CASI using CUDA on the graphics-processing unit (GPU).
  - Assisted with generating marketing plan, business plan, and licensing agreements with the six other employees of TerraSpark Geosciences.

- Student Visitor – National Center for Atmospheric Research (May 2005 - Present)
  - Implemented domain-decomposition techniques in C++ with MPI and OpenMP for examining performance and scalability of the Sea Ice component of the Community Climate System Model on the IBM Blue Gene/L supercomputer.

- Research Assistant II – Minnesota Supercomputing Institute (Sept 2002 – May 2005)
  - Developed parallel C++ algorithm for clustering 3D earthquake events in time and space, which has been the basis for 6 publications.
  - Developed with two other students the WEB-IS system for remote visualization and analysis of large datasets using Java, OpenGL, Python, and SOAP.
  - Lead project to explore the use of web cam technologies to enable remote visualization of large datasets, collaboration between researchers, and disaster information dissemination.

Past Education:
- MS in Computer Science, University of Colorado (Dec. 2006)
- BS in Computer Engineering, Senior Honors, Univ. of Minnesota (Sept. 2001 - May 2005)
- Saint Mary’s Central High School, Bismarck, ND (1997-2001)

Skills:
- Proficient in C++, C, CUDA, Java, Fortran 90, Perl, OpenGL 2.0, GLSL.
- Experience in Windows, Linux and Macintosh environments.
- Experience in graphics, visualization, geometric, modeling, and mathematical algorithms
- Experience in multi-threaded, parallel, and GPU programming.

Honors:
- Three Provisional Patent Applications in Seismic Interpretation, Jan 2007 to Present.
- Visiting student to Supercomputing Center at Chinese Academy of Sciences, June 2007.
- National Science Foundation Graduate Research Fellowship Honorable Mention, 2006.
- Univ. Minnesota Computer Engineering Honors Program and Boeing Scholarship 2004/2005
- Undergraduate Research Opportunities Program (UROP) Individual Award Recipient, 2004
- Visitor to Chinese Earthquake Admin. and Changbaishan Volcano Observatory, Jul. 2004

Publications and Abstracts (Uncategorized Chronological List)

Kadlec, B.J., Tufo, H.M. Visualization and Computational Steering of Level Sets on the GPU for Seismic Interpretation, 10th LCI Conference on High Performance Clustered Computing, March 2009 (poster).


Yuen, D.A., Dzwiel, W., Ben-Zion, Y., Boryczko, K., Kadlec, B.J., Bollig, E.F., Yoshioka, S., Erlebacher, G., Clustering Analysis, Data-Mining, Visualization of Earthquakes over the Internet, APEC Cooperation for Earthquake Simulation (ACES), July 9-14, 2004, Beijing, China.


