SEISMIC VOLUMETRIC FLATTENING AND SEGMENTATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF GEOPHYSICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Jesse Lomask
June 2007

Printed as Stanford Exploration Project No. 126
by permission of the author

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Jon F. Claerbout) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Biondo Biondi)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Norman Sleep)

Approved for the University Committee on Graduate Studies.

Abstract

The use of computers in 3-D seismic volume interpretation has not reached its full potential. Modern interpreters have many tools at their disposal including horizon and fault auto-trackers and 3-D volume interpretation software packages. However, even with these tools, interpreters most often resort to picking on 2-D slices. This is because humans are most proficient at interpreting 2-D images. In 2-D, they are able see complicated patterns and trends in somewhat noisy data. Computers, on the other hand, are not limited to 2-D. Computers can take advantage of the full dimensionality of the data and offer solutions based on all of the data. One particular interpretation technique that would benefit from the computer using the full dimensionality of the data is to flatten the data into horizon slices. This is referred as, in general, flattening. Another technique is to use image segmentation to pick salt boundaries via a globally optimized problem. That is to cluster seismic data cubes into two groups. One group is inside the salt and one group is outside the salt. Where the two groups meets is the salt boundary.

Towards that end, I present an efficient full-volume automatic dense-picking method for flattening seismic data. First local dips (step-outs) are calculated over the entire seismic volume. The dips are then resolved into time shifts (or depth shifts) using a non-linear Gauss-Newton iterative approach that exploits Discrete Cosine Transforms (DCTs) to minimize computation time. To handle faults (discontinuous reflections), I apply a weighted inversion scheme. The weight identifies locations of faults allowing dips to be summed around the faults in order to reduce the influence of erroneous dip estimates in the vicinity of the fault. If a fault model is not provided, I can estimate a suitable weight (essentially a fault indicator) within the inversion using iteratively re-weighted least-squares (IRLS). The method is tested

successfully on both synthetic and field data sets of varying degrees of complexity including salt piercements, angular unconformities, and laterally limited faults. In areas with faults and poor signal/noise ratio, where reflectors can be discontinuous from place to place, a dip-based flattening technique might not be able to appropriately track sedimentary layers. To aid the flattening algorithm, one or a few reflectors can be picked. This information can be then incorporated in the algorithm as geological constraints. I add a model mask to a time domain solution using a Gauss-Newton approach that incorporates an initial solution. Having incorporated the geological information, the flattening algorithms can accurately pick reflectors in 3D for a noisy field data example. This method is also able to pick across faults requiring a minimal amount of interpretation.

A second full-volume interpretation method uses normalized cuts image segmentation to track salt interfaces. I apply a modified version of the normalized cuts image segmentation (NCIS) method to partition seismic images along salt interfaces. The method is capable of tracking interfaces that are not continuous, where conventional horizon tracking algorithms may fail. NCIS calculates a weight connecting each voxel in the image to each voxel in a local neighborhood. In my implementation, weights are determined from a problem-dependent combination of attributes, the most important attributes being instantaneous amplitude and dip. The weights for the entire image are placed in a weight matrix and used to segment the image via an eigenvector calculation. The weight matricies for 3D seismic data cubes can be quite large and computationally expensive. By imposing bounds and by distributing the algorithm on a parallel cluster, I significantly increase efficiency and robustness. This method is demonstrated to be effective on both 2D and 3D seismic data sets.

Preface

This thesis is essentially a compilation of two main projects of research aimed at full volume interpretation: flattening and segmentation. Because each of these research projects is actually a collaborative effort of several authors, "I" actually means "we" throughout this document. For flattening, the primary collaborating authors are Jon Claerbout, Antoine Guitton, and Sergey Fomel. For segmentation, the primary collaborating authors are Biondo Biondi and Robert Clapp.

All of the figures in this thesis are marked with one of the three labels: [ER], [CR], and [NR]. These labels define to what degree the figure is reproducible from the data directory, source code and parameter files provided on the web version of this thesis [1].

ER denotes Easily Reproducible. The author claims that you can reproduce such a figure from the programs, parameters, and data included in the electronic document. We assume you have a UNIX workstation with Fortran90, Fortran77, C, X-Window system, and the SEPlib software package. Before the publication of the electronic document, someone other than the author tests the author's claim by destroying and rebuilding all ER figures.

CR denotes Conditional Reproducibility. The author certifies that the commands are in place to reproduce the figure if certain resources are available. SEP staff have not attempted to verify the author's certification. To find out what the required resources are, you can inspect a corresponding warning file in the document's result directory. For example,

[1] http://sepwww.stanford.edu/public/docs/sep10?

you might need a large or proprietary data set. You may also need a super computer, or you might simply need a large amount of time (20 minutes or more) on a workstation.

NR denotes Non-Reproducible. This class of figure is considered non-reproducible. Figures in this class are scans and artists' drawings. Output of interactive processing is labeled NR.

Acknowledgements

First and foremost, I thank Mike Lomask. He was a great brother and friend, who accomplished so much in just a short time. He continues to impact my life. I also thank the other members my family particularly Jodi, Marisa, Joe, Mort, Grant, Tristan, Ravi, Joan, and Mort.

Jon Claerbout has been a great advisor in many ways. After several years, I have just scratched the surface of what I could learn from him. He has tremendously enhanced how I look at geophysical problems. I also thank Biondo Biondi and Bob Clapp for making the SEP experience a healthy balance of fun and productivity. There are too many others in the SEP community to thank individually. The memories of strolling to the book store to get coffee will always be good. Key friends and collaborators include Antoine Guitton, Daniel Rosales, Guojian Shan, Alejandro Valenciano, Bill Curry, Gabriel Alvarez, and Morgan Brown.

I thank Brian Bergquist, Chevron, Spirit 76 - Unocal, and Total for supplying the data used in this document. I thank James Rickett and Dave Hale for many useful suggestions. I also thank Moritz Fleidner and 3DGEO Development Incorporated for processing some of the data. I would also like to acknowledge the financial support of the sponsors of the Stanford Exploration Project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In spite of numerous advances in computational power in recent years, seismic interpretation still requires significant manual picking. The most obvious explanation of this is that interpretation involves a lot of pattern recognition, a task that humans excel at in two dimensions. Beyond 2D, a human gets quickly lost. On the other hand, today's computers are easily able to handle higher dimensionality; they are able find optimized solutions for large problems that are consistent across many dimensions simultaneously. Unfortunately, many of the available computerized tools fail to fully exploit the full dimensionality of the data. As a result, much of seismic data is interpreted one 2D slice at a time.

The time is right for the seismic industry to address this problem by increasing the role of the computer in seismic interpretation. Many forces are in motion throwing computerized interpretation to the forefront of technological development. The current high demand for oil coupled with increased retirements is straining the interpretation workforce. This results in fewer interpreters with greater amounts of data to interpret. Add to this the need to go after more complicated stratigraphic plays as the simple structural plays are diminishing and the improved image capabilities create higher quality datasets that have more to offer interpretively than ever before. All of these forces will create an environment in the coming years, that will likely see great development in technologies where computers will take a greater role in interpretation and the human will be able to focus more on the challenging areas.

Manual interpretation of 3D seismic data sets is generally carried out by extraction of 2D sections in loops. The loops are closed so that the beginning and end are coincident. Horizons or events are tracked along these sections from point to point. If the beginning of a tracked horizon does not match the end, then a mis-tie (error in the tracking) has occured. This process is referred to as loop-tying. Viewing a loop-tied section, the interpreter can see errors or mis-ties and then make adjustments and then select another loop. By repeating this iterative process of looking at many 2D sections, gradually the full 3D volume is interpreted. Even though the human is easily able to see patterns in the 2D line, only one 2D section of a 3D cube is used at a time. In Figure 1.1 is a cartoon illustrating the process of loop-tying. Every time the human

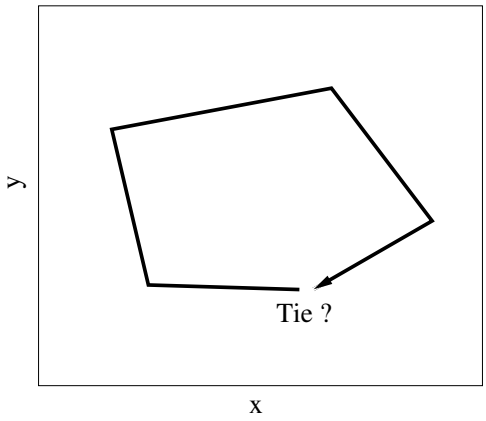

Figure 1.1: A map view cartoon illustrating the process of loop-tying. In this process, an interpreter starts picking on a 2D section along a circular loop. A mis-tie between the starting and ending pick is evidence of errors somewhere along the loop. intro-looptie [NR]

gets back to the starting point, mis-ties can occur. These mis-ties only indicate that an error in picking has occurred somewhere along the loop.

On the other hand, certain visualization products and autopickers seek to make picking processes as efficient as possible. However, they often suffer from weaknesses that prevent them from being truly practical. For example, 3D volume interpretation packages allow interpreters to view their data with depth perception using stereo glasses. These products have an opacity ability (James et al., 2002) that allows interpreters to make unwanted data transparent. Unfortunately, unless the zone of interest has a known unique range of attribute values, interpreters resort to picking on 2-D slices. Similarly, computerized autopickers suffer from deficiencies as well. Unlike manual loop-tying work-flows, seed-based autotrackers begin tracking in all directions from a seed point. Following an often complicated set of rules, the surface is tracked from point to point by solving local problems. Where two paths converge,

mis-ties can occur, again merely indicating an error had occurred somewhere along either of the two paths. This is drawn in Figure 1.2. Like manual loop-tying, autotrackers only use a small piece of the data at time. Even though the auto-tracker is able to track outward in all directions from a seed point at once, it is still only solving a series of local problems, thus only seeing the trees and not the forest. Additionally, traditional amplitude-based autopickers can fail if the horizon being tracked has significant amplitude variation, or worse, polarity reversal. Other tracking techniques such as artificial neural networks are less sensitive to amplitude variations but are still prone to error if the seismic wavelet character varies significantly from the training data (Leggett et al., 1996).

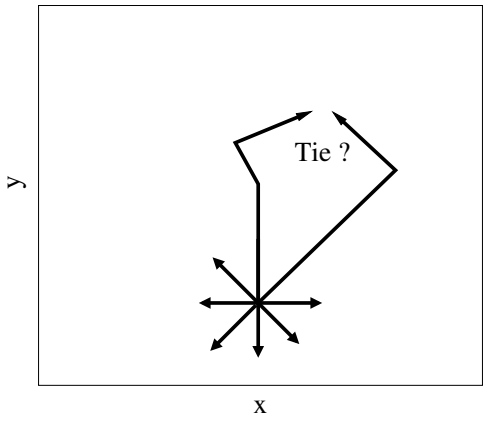

Figure 1.2: A map view cartoon illustrating the behavior of an autotracker with a seed point. Mis-ties can occur at the intersection of two tracking paths. intro-1seed [NR]

In this thesis, I describe two technologies recently developed to bring more of the computer's ability to handle multiple dimensions to bear on the problem of seismic data interpretation. The aim of both, is so use more of the data simultaneously to solve full-volume problems. Towards this end, these methods should allow the computer to see more of the forest instead of just the trees. The first is volumetric flattening and the second is image segmentation for tracking salt boundaries.

Volumetric flattening

In Chapter 2, I present a novel efficient full-volume automatic dense-picking method for flattening seismic data. First local dips (step-outs) are calculated over the entire seismic volume. The dips are then resolved into time shifts (or depth shifts) relative to reference trace using a

non-linear Gauss-Newton iterative approach that exploits Discrete Cosine Transforms (DCT's) to minimize computation time. At each point in the image two dips are estimated; one dip in the x direction and one dip in the y direction. Because each point in the image has two dips, each horizon is estimated from an over-determined system of dips in a least-squares sense. This is akin to having the computer automatically loop-tie every possible loop simultaneously. Subsequently, each trace is shifted according to the time shifts to match the reference trace. To further exploit the dimensionality of the data, all of the horizons are estimated at once in 3D to conform to one another. On the left of Figure 1.3 is a 3D salt piercement data cube. On

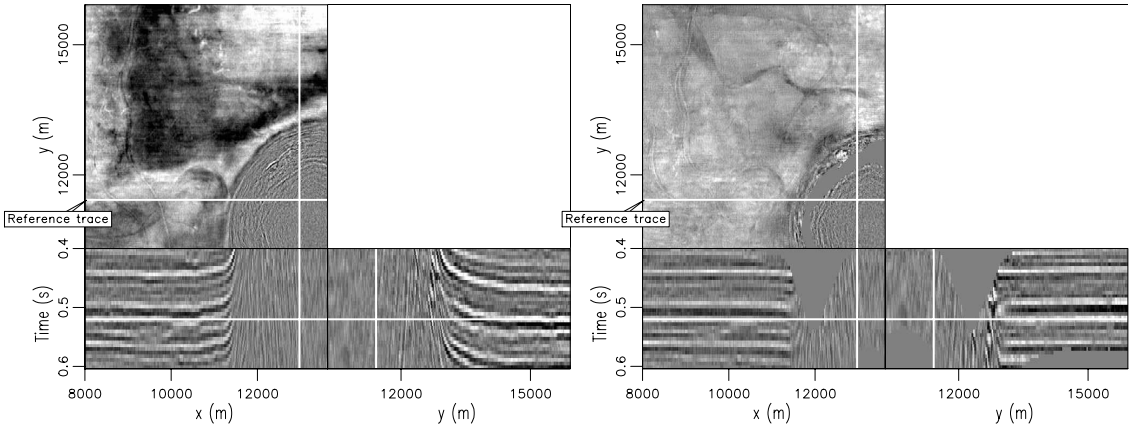

Figure 1.3: The left side is an input data cube. The right side is the flattened result. It has been converted to a cube of horizon slices. intro-flat_merge [NR]

the right is a flattening result. The structure from the image on the left was removed without any human interpretation. An interpreter can now easily glance throught the cube of horizon slices and see channels and other geological features as they were laid down. The computer found this result efficiently using all of the data in a full volume least-squares sense.

To handle faults (discontinuous reflections), I apply a weighted inversion scheme. The weight identifies locations of faults allowing dips to be summed around the faults in order to reduce the influence of erroneous dip estimates in the vicinity of the fault. If a fault model is not provided, I can estimate a suitable weight (essentially a fault indicator) within the inversion using iteratively re-weighted least-squares (IRLS). The method is tested successfully on

both synthetic and field data sets of varying degrees of complexity including salt piercements, angular unconformities, and laterally limited faults.

In areas with faults and poor signal/noise ratio, where reflectors can be discontinuous from place to place, a dip-based flattening technique might not be able to appropriately track sedimentary layers. In such cases, flattening results are improved by adding constraints. The constraints can be individual picks, entire picked horizons, correlations across faults, or well ties. I implement constraints by passing a binary model mask and initial solution to a Gauss-Newton flattening approach that uses the DCT method as a preconditioner. Having included some geological information, I demonstrate that the flattening algorithm can accurately pick reflectors in 3D for both faulted and noisy field data examples.

Image segmentation for tracking salt boundaries

In Chapter 3, I describe the application of normalized cuts image segmentation (Shi and Malik, 2000) to the problem of picking 3D salt boundaries. This unique picking method solves a global optimization problem to estimate a picked salt boundary. This method is less prone to errors from local amplitude fluctuations and discontinuities than previous picking methods. The method essentially clusters the data cube into two groups. One group is inside the salt body, whereas the other group is outside the salt body. Where the two groups meet is the picked salt interface. An sample result is shown in Figure 1.4.

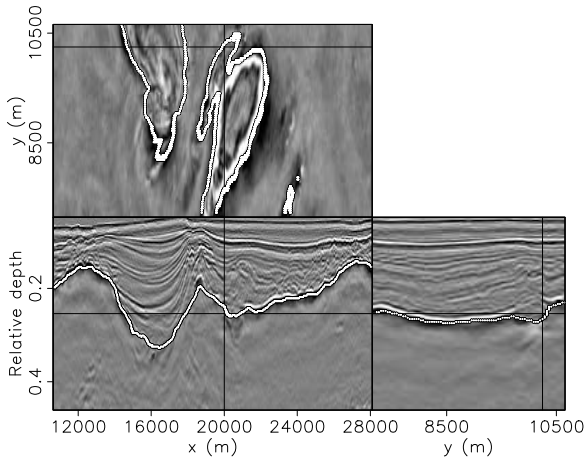

Figure 1.4: A salt boundary that has been tracked using normalized cuts image segmentation. intro-combo_pck1_win_thick2_sm [NR]

This method operates in a seismic data cube by creating weights relating each voxel to every voxel in its neighborhood. These weights are then placed in a weight matrix and used to find an optimum clustering partition via an eigenvector problem. Implicitly built into this method is a measure of picking uncertainty. Additionally, bound-constraints can be imposed to incorporate interpreter influence into the result and (more importantly) to reduce the significant computational requirements of this method. To further increase practicality of this method, I distribute two key steps on a parallel network. Examples, both 2D and 3D, demonstrate its use.

Chapter 2

Flattening

INTRODUCTION

One of the main goals of interpretation is to extract geological and reservoir features from seismic data . One commonly used interpretation technique that helps with this effort is flattening data on horizons [e.g. Lee (2001)], also known as stratal slicing (Zeng et al., 1998). This procedure removes structure and allows the interpreter to see geological features as they were emplaced. For instance, after flattening seismic data, an interpreter can see in one image an entire flood plain complete with meandering channels. However, in order to flatten seismic data, a horizon needs to be identified and tracked throughout the data volume. If the structure changes often with depth, then many horizons need to be identified and tracked. This picking process is time consuming and expensive.

In this chapter, I demonstrate a method for automatically flattening entire 3D seismic cubes without manual picking. This concept was initially presented by Lomask and Claerbout (2002). This is an efficient algorithm that intrinsically performs automatic dense picking on entire 3D cubes at once. The method involves first calculating local dips (step-outs) everywhere in the data using a dip estimation technique (Claerbout, 1992; Fomel, 2002). These dips are resolved into time shifts (or depth shifts) via a non-linear least-squares problem. I use an iterative Gauss-Newton approach that integrates dips quickly using discrete cosine transforms

(DCTs). The data are subsequently vertically shifted according to the summed time shifts to output a flattened volume. Bienati and Spagnolini (1998), Bienati et al. (1999a,b), and Bienati and Spagnolini (2001) use a similar approach to numerically resolve the dips into time shifts for the purpose of auto-picking horizons and flattening gathers, but solve a linear version of the problem. Stark (2004) takes a full-volume approach to achieve the same goal but unwraps instantaneous phase instead of dips. Blinov and Petrou (2003) use dynamic programming to track horizons by summing local dips.

As with amplitude-based autopickers, amplitude variation also affects the quality of the dip estimation and will, in turn, impact the quality of the flattening method presented here. However, the effect will be less significant because the method can flatten the entire data cube at once, globally, in a least-squares sense, minimizing the effect of locally poor dip information.

Discontinuities in the data from faults tend to corrupt the local dip estimates at the faults. In this case, weights identifying the faults are applied within the iterative scheme, allowing reconstruction of horizons for certain fault geometries. Such weights could be obtained from a previously determined fault model. If a fault model is not provided, I can automatically generate suitable weights using iteratively re-weighted least squares (IRLS). Once a seismic volume is flattened, automatic horizon tracking becomes a trivial matter of reversing the flattening process to unflatten flat surfaces.

Although unconstrained flattening finds a global least-squares solution, if there is enough noise, the flattening result will have significant errors. Similarly, even though flattening can reconstruct horizons with certain fault geometries, there are numerous geometries that cannot be reconstructed. These issues can be alleviated by introducing hard constraints into the flattening algorithm. My flattening method has the ability to integrate some manual interpretation into its solution as constraints. Furthermore, the use of constraints opens the door on iterative flattening: an approach where the flattening result is improved by iteratively adding constraints.

The applications of flattening are numerous. The flattening transformation captures all of the information about the shape of the structure and, in some cases, the stratigraphy, of the

data. This leads to potential applications that include understanding stress/strain relationships, timing of events, and facies pattern recognition. The prestack applications for this method are abundant too. For example, time shifts can easily be incorporated into an automatic lateral velocity-estimating scheme (Guitton et al., 2004).

In the following sections, I present the flattening methodology and a series of real-world geological challenges for this method. The first is a 3D synthetic data set generated from a model with a dipping fault and thinning beds. Then I present a structurally simple, 3D salt-piercement field data set from the Gulf of Mexico. I consider it structurally simple because the geological dips do not change significantly with depth. Increasing complexity, next I flatten a 3D field data set from the North Sea that contains an angular unconformity and has significant folding. Next, I present a 3D faulted field data set from the Gulf of Mexico. Then I describe flattening with geological constraints. I illustrate its use with two 3D field data examples. The first is a faulted data set and the second is a noisy one. Lastly, I discuss several additional flattening applications such as automatic generation of fault-slip distributions, stress volumes, and unconformity volumes.

FLATTENING THEORY

Ordinarily, in order to flatten a single surface, each sample is shifted vertically to match a chosen reference point. For instance, this reference point can be the intersection of the horizon and a well pick. For multiple horizons, this reference point becomes a vertical line (reference trace). In Figure 2.1 is a cartoon of a horizon and the mapping field represented by arrows. In order to flatten 3D cubes, the objective is to find a mapping field $\boldsymbol{\tau}(x,y,t)$ such that each time slice of this field contains the locations of all the data points along the horizon that happens to intersect the reference trace at that time slice. The $\boldsymbol{\tau}$ field is found by summing dips. The basic idea is similar to phase unwrapping (Ghiglia and Romero, 1994), but instead of summing phase differences to get total phase, dips are summed to get total time shifts that are then used to flatten the data.

The first step is to calculate local dips everywhere in the 3D seismic cube. Dips can be efficiently calculated using a local plane-wave destructor filter as described by Claerbout

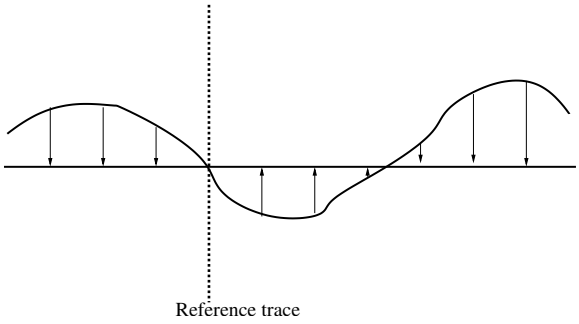

Figure 2.1: A cartoon illustrating the flattening of a horizon. The arrows represent the $\boldsymbol{\tau}$ field that provide a mapping from the reflector to the flattened horizon at the intersection of the reflector and the reference trace. flat-tau_example [NR]

(1992) or with an improved dip estimator that is described by Fomel (2002). I primarily use the latter. For each point in the data cube, two components of dip, **b** and **q**, are estimated in the x and y directions, respectively. These can be represented everywhere on the mesh as $\mathbf{b}(x,y,t)$ and $\mathbf{q}(x,y,t)$. If the data to be flattened is in depth, then dip is dimensionless, but in the case the data is in time, dip has units of time over distance.

A simple flattening method is to simply take each time slice of the estimated dips and sum them into a total time-shift (or depth-shift) field $\boldsymbol{\tau}(x,y,t)$. In other words, for each time slice of $\mathbf{b}(x,y,t)$ and $\mathbf{q}(x,y,t)$, I solve a linear least-squares problem. Using the gradient operator ($\boldsymbol{\nabla} = [\frac{\partial}{\partial x} \ \frac{\partial}{\partial y}]^{\mathrm{T}}$) and the estimated dip ($\mathbf{p} = [\mathbf{b} \ \mathbf{q}]^{\mathrm{T}}$), I wish to find a $\boldsymbol{\tau}(x,y,t)$ field such that its gradient approximates the over-determined system of dips using the fitting goal

$$\mathbf{0} \quad \approx \quad \text{residual} \quad = \quad \boldsymbol{\nabla}\boldsymbol{\tau}(x,y,t) - \mathbf{p}(x,y,t). \tag{2.1}$$

The linear least-squares solution to this flattening problem is

$$\boldsymbol{\tau} \quad = \quad (\boldsymbol{\nabla}^{\mathrm{T}}\boldsymbol{\nabla})^{-1}\boldsymbol{\nabla}^{\mathrm{T}}\mathbf{p}. \tag{2.2}$$

The resulting time-shift field $\boldsymbol{\tau}(x,y,t)$ contains all of the time shifts required to map the original unflattened data to flattened data. This is implemented by applying the time shifts relative to a reference trace. In other words, each trace is shifted to match the reference trace. For simplicity, I assume the reference trace to be a vertical line, however it could, in principle, be non-vertical or even discontinuous.

I can solve this problem efficiently in the Fourier domain using the Fast Fourier Transform.

I apply the divergence to the estimated dips and divide by the z-transform of the discretized Laplacian (Lomask and Claerbout, 2002) in the Fourier domain, i.e.,

$$\boldsymbol{\tau} \quad \approx \quad \mathrm{FFT_{2D}}^{-1}\left[\frac{\mathrm{FFT_{2D}}\left[\nabla^{\mathrm{T}}\mathbf{p}\right]}{-Z_x^{-1}-Z_y^{-1}+4-Z_x-Z_y}\right] \tag{2.3}$$

where $Z_x = e^{iw\Delta x}$ and $Z_y = e^{iw\Delta y}$. This Fast Fourier approach is similar to the method of Ghiglia and Romero (1994) for unwrapping phase. This requires calling both a forward and inverse FFT. The ability to invert the 2D Laplacian in one step is the key to this method's computational efficiency. To avoid artifacts in the Fourier domain, I could mirror the divergence of the residual $\nabla^{\mathrm{T}}\mathbf{r}$ before I apply the Fourier transform. Mirroring is described by Ghiglia and Pritt (1998) for application to 2D phase unwrapping. Unfortunately, this requires a factor of 4 increase in memory in 2D and a factor of 8 in 3D. By using the discrete cosine transform (DCT), also described by Ghiglia and Romero (1994), I eliminate the need for mirroring. This cosine transform method is

$$\boldsymbol{\tau} \approx \mathrm{DCT_{2D}}^{-1}\left[\frac{\mathrm{DCT_{2D}}\left[\nabla^{\mathrm{T}}\mathbf{p}\right]}{J}\right], \tag{2.4}$$

where $\mathrm{DCT_{2D}}$ is the 2D discrete cosine transform and

$$J = -2(\cos(w\Delta x)+\cos(w\Delta y))+4. \tag{2.5}$$

J is the real component of the Z-transform of the 2D finite difference approximation to the Laplacian.

By fitting the integrated time-shift field to the time slices of the dip field, I am assuming that the structure does not vary with time (or depth). Although, there are many seismic data examples where the structure changes slowly enough to allow this assumption, it is more desirable, in general, to fit the $\boldsymbol{\tau}$ field not to the dips along each times slice but instead along each horizon. In other words, the goal is to find a time-shift (or depth-shift) field $\boldsymbol{\tau}(x,y,t)$ such that its gradient approximates the dip $\mathbf{p}(\mathrm{x},\mathrm{y},\boldsymbol{\tau})$. The dip is now a function of $\boldsymbol{\tau}$ because for any given horizon, the appropriate dips to be summed are the dips along the horizon itself.

The regression (developed by Sergey Fomel in Appendix A) now is

$$\mathbf{0} \quad \approx \quad \text{residual} \quad = \quad \nabla\boldsymbol{\tau}(x,y,t) - \mathbf{p}(x,y,\boldsymbol{\tau}) \tag{2.6}$$

Note that because the estimated dip $\mathbf{p}(\mathrm{x},\mathrm{y},\boldsymbol{\tau})$ field is a function of the unknown $\boldsymbol{\tau}(x,y,t)$ field, this problem is non-linear and, therefore, difficult to solve directly. I solve it using a Gauss-Newton approach by iterating over the following equations:

iterate {

$$\mathbf{r} \quad = \quad \nabla\boldsymbol{\tau}_k(x,y,t) - \mathbf{p}(x,y,\boldsymbol{\tau}_k) \tag{2.7}$$

$$\Delta\boldsymbol{\tau} \quad = \quad (\nabla^{\mathrm{T}}\nabla)^{-1}\nabla^{\mathrm{T}}\mathbf{r} \tag{2.8}$$

$$\boldsymbol{\tau}_{k+1}(x,y,t) \quad = \quad \boldsymbol{\tau}_k(x,y,t) + \Delta\boldsymbol{\tau} \tag{2.9}$$

} ,

where the subscript k denotes the iteration number.

A more intuitive way to understand this method is to consider the first iteration. If no initial solution is provided, $\boldsymbol{\tau}_0$ will be zero and the residual $\mathbf{r}$ will be the input dips $\mathbf{p}$ along each time slice. These dips are then summed into time shifts using equation (2.8). This is the linear least-squares solution in equation (2.2). At this point, these time shifts can be used to flatten the data but because the dips were summed along time slices and not along individual horizons, the data will not be perfectly flat. The degree that it is not flat is related to the variability of the dip in time. At this point, I could re-estimate new dips on the partially flattened data and then repeat the process. In general, iterating in this way will eventually flatten the data. This is essentially how this method works, but instead of re-estimating dips at each iteration, I correct the original dips each iteration with equation (2.7). This is not only more efficient than estimating new dips at each iteration but also more robust as discontinuities introduced by flattening do create inaccurate dips.

Still, I leverage the efficiency of the Fourier domain by solving equation (2.8) using the

same cosine transform method as equation (2.4) as

$$\Delta\boldsymbol{\tau} \approx \mathrm{DCT_{2D}}^{-1}\left[\frac{\mathrm{DCT_{2D}}\left[\nabla^{\mathrm{T}}\mathbf{r}\right]}{J}\right], \tag{2.10}$$

where $\mathrm{DCT_{2D}}$ is the 2D discrete cosine transform and

$$J = -2(\cos(w\Delta x)+\cos(w\Delta y))+4. \tag{2.11}$$

J is the real component of the Z-transform of the 2D finite difference approximation to the Laplacian. Each iteration requires a forward and inverse DCT.

Convergence is generally reached when the change in normalized residual between consecutive iterations is smaller than a user specified tolerance μ, i.e. ,

$$\left|\frac{\|\mathbf{r}_{k-1}\|-\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|}\right| \quad < \quad \mu. \tag{2.12}$$

I have found that a value $\mu = 0.001$ often gives sufficiently flat results.

An alternative stopping criterion is to consider only the norm of the residual $\|\mathbf{r}_k\|$ because it is essentially the sum of the remaining dips. In this case, the stopping tolerance would be the user specified minimum average dip value. In other words, convergence will be achieved when the average dip of the flattened volume is below a user specified threshold. Unfortunately, this is very sensitive to the amount of noise and, consequently, would not make a satisfactory stopping criterion. The stopping criterion is in equation (2.12) is less sensitive to the absolute value of the noise because it is considering only the change in residual from iteration to iteration.

Once convergence has been achieved, the resulting time-shift field $\boldsymbol{\tau}(x,y,t)$ is used to map the data cube into a cube of horizon slices. In general, I operate on one time slice at a time. After iterating until convergence, I then select the next time slice and proceed down through the cube. In this way, each time slice is solved independently. However, the number of iterations can be significantly reduced by passing the solution of the previous time slice as an initial solution to the current time slice.

The process of flattening will stretch or compress the data in time, hence altering the

spectrum. Even worse, it can disrupt its continuity. To ensure a monotonic and continuous result, it should help to smooth the input dips in time (or depth). In some instances, it may be necessary to enforce smoothness during the integration of the dips. It helps to use as much of the data as possible when finding a flattening solution. This can be accomplished by adding regularization in the time direction. To accomplish this, I apply a 3D gradient operator with a residual weight $\mathbf{W}_\epsilon$ that controls the amount of vertical regularization defined as

$$\mathbf{W}_\epsilon \boldsymbol{\nabla} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\epsilon} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial t} \end{bmatrix} \quad , \tag{2.13}$$

where $\mathbf{W}_\epsilon$ is a large block diagonal matrix consisting of two identity matrices $\mathbf{I}$ and a diagonal matrix $\boldsymbol{\epsilon}$=$\epsilon\mathbf{I}$.

For simplicity, I merge this weight operator to the gradient operator to create a new operator now defined as a 3D gradient with an adjustable weighting parameter ϵ as

$$\boldsymbol{\nabla}_\epsilon \quad = \quad \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \epsilon \frac{\partial}{\partial t} \end{bmatrix} \quad . \tag{2.14}$$

The residual is now defined as

$$\mathbf{r} \quad = \quad \boldsymbol{\nabla}_\epsilon \boldsymbol{\tau} - \mathbf{p}_\epsilon \quad = \quad \begin{bmatrix} \frac{\partial \boldsymbol{\tau}}{\partial x} \\ \frac{\partial \boldsymbol{\tau}}{\partial y} \\ \epsilon \frac{\partial \boldsymbol{\tau}}{\partial t} \end{bmatrix} \quad - \quad \begin{bmatrix} \mathbf{b} \\ \mathbf{q} \\ \mathbf{0} \end{bmatrix}. \tag{2.15}$$

The new operator $\boldsymbol{\nabla}_\epsilon$ has a scalar parameter ϵ used to control the amount of vertical smoothing. In the case of flattening an image in depth, ϵ is dimensionless and in the case of time, it has units of time over distance. This operator requires integrating the entire 3D volume of dips at once rather than slice by slice.

For 3D, I solve this using a Gauss-Newton approach by iterating over equations (2.16)-(2.18), i.e.,

iterate {

$$\mathbf{r} = [\nabla_\epsilon \boldsymbol{\tau}_k(x,y,t) - \mathbf{p}_\epsilon(x,y,\boldsymbol{\tau}_k)] \tag{2.16}$$

$$\Delta\boldsymbol{\tau} = (\nabla_\epsilon^{\mathrm{T}}\nabla_\epsilon)^{-1}\nabla_\epsilon^{\mathrm{T}}\mathbf{r} \tag{2.17}$$

$$\boldsymbol{\tau}_{k+1}(x,y,t) = \boldsymbol{\tau}_k(x,y,t) + \Delta\boldsymbol{\tau} \tag{2.18}$$

} , where the subscript k denotes the iteration number.

The 2D DCTs in equation (2.10) are replaced with 3D DCTs. Consequently, each iteration is slowed. I solve equation (2.17) using a 3D version of a 2D ((x,y)-plane) method described by Ghiglia and Pritt (1998) (for phase unwrapping) as

$$\Delta\boldsymbol{\tau} \approx {\mathrm{DCT}_{3\mathrm{D}}}^{-1}\left[\frac{\mathrm{DCT}_{3\mathrm{D}}\left[{\nabla_\epsilon}^{\mathrm{T}}\mathbf{r}\right]}{J}\right], \tag{2.19}$$

where $\mathrm{DCT}_{3\mathrm{D}}$ is the 3D discrete cosine transform and

$$J = -2(\cos(w\Delta x) + \cos(w\Delta y)) + 2\epsilon^2(1-\cos(w\Delta t)) + 4. \tag{2.20}$$

J is the real component of the Z-transform of the 3D finite difference approximation to the Laplacian with adjustable regularization.

The magnitude of the smoothing parameter ϵ depends on competing requirements of the amount of noise and structural complexity. If the structure is changing considerably with time, then it should be chosen to be as small as possible. However, if there is significant noise resulting in erroneous dips then it should be larger. When ϵ=1, the inversion honors the dips and smoothness criteria equally.

There is a subtle difference between smoothing the input dips or smoothing the time-shift field. The time shift is essentially an integration of the input dips, so small errors in the input dips can accumulate into large errors in the cumulative time-shift field. This can be illustrated by describing the simplest non-linear flattening method applied to (x,t)-plane vertical sections without least-squares:

iterate {

$$\begin{aligned} \mathbf{r} &= \nabla\boldsymbol{\tau}_k(x,t) - \mathbf{b}(x,\boldsymbol{\tau}_k) && (2.21)\\ \Delta\boldsymbol{\tau} &= \nabla^{-1}\mathbf{r} && (2.22)\\ \boldsymbol{\tau}_{k+1}(x,t) &= \boldsymbol{\tau}_k(x,t) + \Delta\boldsymbol{\tau} && (2.23) \end{aligned}$$

} ,

where the subscript k denotes the iteration number and $\mathbf{b}(x,\boldsymbol{\tau})$ is the dip in the x direction. Equation (2.22) can easily be solved with 1D causal integration. Because of the nonlinearity, smoothing the input dips $\mathbf{b}(x,\boldsymbol{\tau})$ is not equivalent to smoothing the resulting $\boldsymbol{\tau}(x,t)$ field. In the same way that traditional amplitude-based autopickers can get off the intended horizon and jump to the wrong horizon by making one error, merely smoothing the input dips can cause this algorithm to flatten the wrong reflector. This is less likely to occur by smoothing the time-shift field.

Weighted solution for faults

At a discontinuity, the dip is infinity and any estimate of dip should not be used. To prevent such meaningless dips from adversely affecting the quality of the flattening, I can sum dips around the faults and ignore the spurious dips at the faults. To do this a weighting is applied to the residual to ignore fitting equations that are affected by the bad dips at the faults. The regression to be solved is now

$$\mathbf{0} \approx \mathbf{W}(\nabla\boldsymbol{\tau}(x,y,t) - \mathbf{p}_\epsilon(x,y,\boldsymbol{\tau})). \quad (2.24)$$

The weighted residual in equation (2.7) is now

$$\mathbf{r} = \mathbf{W}(\nabla\boldsymbol{\tau}(x,y,t) - \mathbf{p}_\epsilon(x,y,\boldsymbol{\tau}_k)). \quad (2.25)$$

Similarly, it follows that equation (2.8) should become

$$\Delta\boldsymbol{\tau} \quad = \quad (\boldsymbol{\nabla}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{W}\boldsymbol{\nabla})^{-1}\boldsymbol{\nabla}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{r}. \tag{2.26}$$

However, because I cannot apply a non-stationary weight in the Fourier domain, I simply replace (2.8) with

$$\Delta\boldsymbol{\tau} \quad = \quad (\boldsymbol{\nabla}^{\mathrm{T}}\boldsymbol{\nabla})^{-1}\boldsymbol{\nabla}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{r}. \tag{2.27}$$

By ignoring those weights, I am still able to use the Cosine Transform method in equation (2.10). Naturally, this means that the Cosine Transform method is not approximating the inverse as well as before causing the algorithm to need more iterations to converge.

Iteratively re-weighted least-squares for fault weights

There is also an option of automatically creating the fault weights from the data cube itself within the inversion. It is known that the ℓ_1 norm is less sensitive than the ℓ_2 norm to spikes in the residual (Claerbout and Muir, 1973). I can take advantage of this to honor dips away from faults and ignore inaccurate dips at faults. In other words, I can automatically find the fault model that best flattens the data. By iteratively recomputing a weight and applying it in equation (2.25), I tend to ignore outlier dip values that occur at faults. By gradually ignoring more and more of the outliers, the solution will no longer satisfying an ℓ_2 (least squares) cost function but another more robust cost function.

I chose a more robust version of the Cauchy function (Claerbout, 1999) because it will tend to create weights that are almost binary, much like a fault indicator:

$$\mathbf{W} = \frac{1}{1+(\mathbf{r})^2/\bar{r}^2}, \tag{2.28}$$

where $\bar{r}$ is an adjustable damping parameter. The standard Cauchy function described in Claerbout (1999) has a square root as:

$$\mathbf{W} = \frac{1}{\sqrt{1+(\mathbf{r})^2/\bar{r}^2}}, \tag{2.29}$$

Removing the square root causes outliers to be ignored more. In other words, in the case of faults, it is desirable to completely ignore dips affected by the faults.

By using a Cauchy weight, I am no longer optimizing a convex function. The primary disadvantage is that there is no unique minimum, thus the final solution depends on the initial model. Although the ℓ_1 weight is convex and more robust than the ℓ_2, it does not make a suitable weight for this application because the faults tend to be too smooth. In short, by using a Cauchy weight, this method depends strongly on the quality of the data, precluding its use on very noisy data; with such data it tends to create "false" faults.

The damping parameter $\bar{r}$ controls the sensitivity to outliers. If $\bar{r}$ is large, then the weight will be closer to 1 everywhere and as a result, almost all of the dips will be honored. If $\bar{r}$ is small, more of the outlier dips will be ignored. If $\bar{r}$ is too small then dips that are not affected by faults and noise can be ignored causing "false" faults to be created. Practice has shown that it is better to start out with a relatively large damping parameter value of $\bar{r} > 2$ and then scaling it by 0.8 when the weight is no longer changing. This generally means scaling it every 5 IRLS iterations.

Although I have been able to demonstrate its effectiveness (see examples), this Iteratively Re-Weighted Least-Squares (IRLS) approach has two significant challenges. The first is it adds considerably more iterations to the flattening process. The second is its sensitivity to noise.

Computational cost

For a data cube with dimensions $n = n_t \times n_x \times n_y$, each iteration requires n_t forward and reverse 2D DCTs. Therefore, in 2D ((x, y)-plane) the number of operations per iteration is about $8n + 2n\log(n_x n_y)$. In 3D, the number of operations per iteration is about $8n + 2n\log(n)$. The number of iterations is a function of the variability of the structure and the degree of weighting. For instance, if the structure is constant with time (or depth), then it will be flattened in one iteration (whereas in this case $\mathbf{p}$ is independent of $\boldsymbol{\tau}$ causing the problem to be linear). On the other hand, if a weight is applied and the structure changes much with depth, I find it takes as many as 100 iterations. The IRLS approach requires many more iterations

because the problem is re-solved each time a new weight is estimated. A typical problem may take ten IRLS iterations causing a ten-fold increase in computation time. However, for 2D ((x, y)-plane) initializing each slice with the solution of the previous slice can greatly reduce the total number of iterations required to converge. The required memory usage for both 2D ((x, y)-plane) and 3D is currently about $9n$ but this number can certainly be reduced with more careful programming.

EXAMPLES OF 3D FLATTENING

Here, I demonstrate this flattening method's efficacy on synthetic and field 3D data sets. We start with a synthetic to illustrate how this method can handle data with faults and folds, and subsequently test the method on several 3D field data sets with varying degrees of structural complexity.

Synthetic Data

Figure 2.2 is a 3D synthetic data set that presents three geological challenges. Firstly, the structure is changing with depth, requiring multiple non-linear iterations. Secondly, there is a pinchout (see annotation) where horizons terminate. Thirdly, a significant fault is present in the middle of the cube. As can be seen on the time slice, the fault is laterally limited and terminates within the data cube, i.e., the tip line of the fault is contained within the data. The tip line is a boundary of a fault that delineates the limit of slip. Since dips estimated at fault discontinuities are, in general, inaccurate and will compromise the quality of the flattening result, I use the IRLS method to iteratively estimate a weight that ignores the inaccurate dips estimated at the fault and honors the dips away from the fault.

The flattening result is shown in Figure 2.3. The location of the reference trace is displayed on the horizon slice. This trace is held constant while the rest of the cube is shifted vertically to match it. Notice how the cube is accurately flattened except in the area of the fault itself. The vertical striations on the horizon slice are interpolation errors. The method is able to flatten this cube without any prior information of the fault location, because the fault is laterally limited

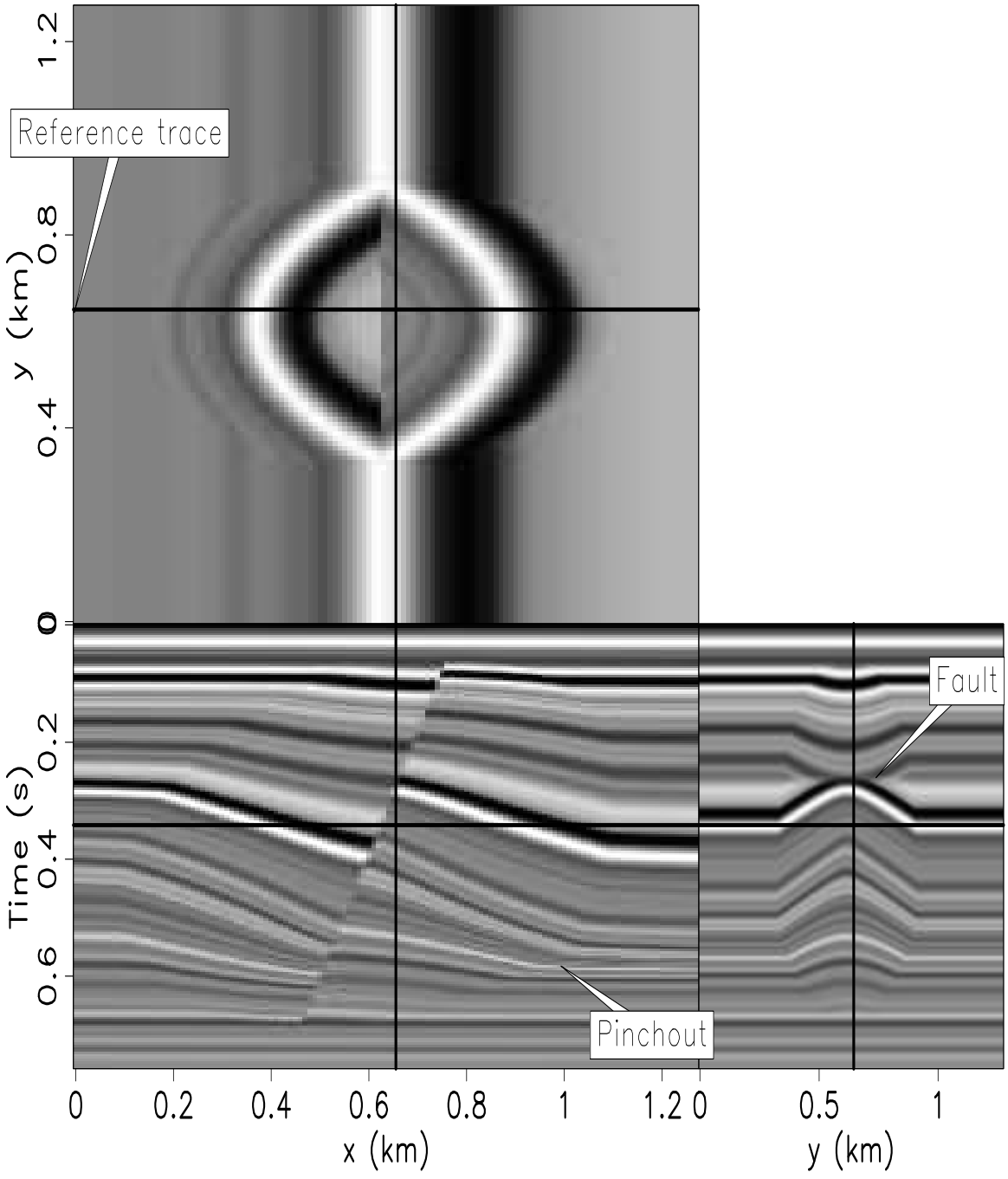

Figure 2.2: A synthetic model with structure varying with depth as well as a dipping discontinuity representing a fault. The black lines superimposed onto the orthogonal sections identify the location of these sections: a time slice at time=0.332 s, an in-line section at y=0.65 km, and a cross-line section at x=0.66 km. The reference trace is located at x=0.0 km and y=0.65 km. flat-downlap [ER]

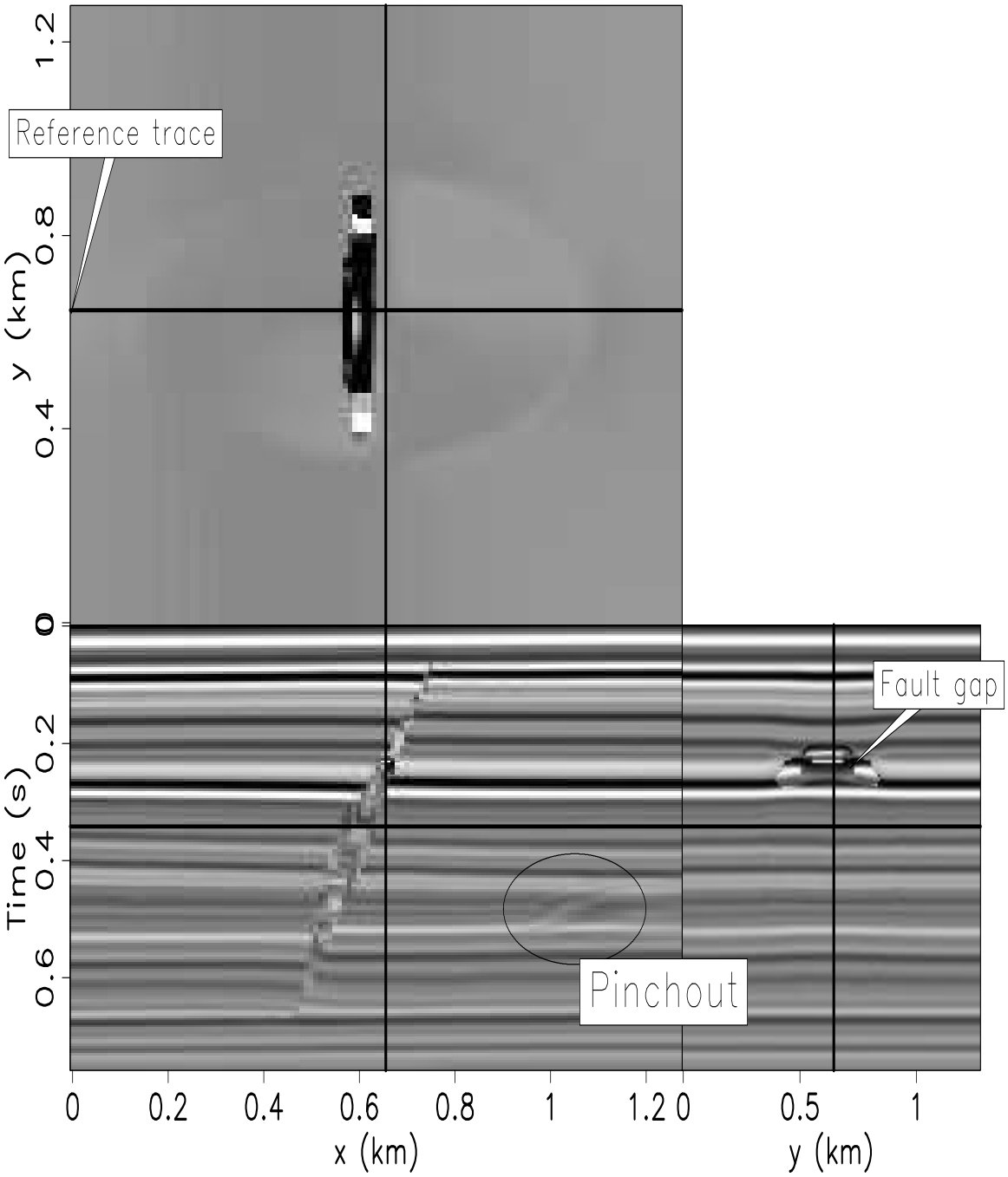

Figure 2.3: As Figure 2.2 only after flattening using IRLS method. Notice how the image is accurately flattened. flat-downlap_flat [ER]

and the signal-to-noise ratio is high. It is important that the fault be laterally limited so that dips can be summed around the fault. Furthermore, the IRLS approach using a Cauchy weight requires a good signal-to-noise ratio to prevent it from creating "false" faults. The term "false" faults refers to the case where the automatically created fault model identifies locations of the data cube as having faults when it only contains some noise. The damping parameter $\bar{r} = 0.2$ was found by testing on one time slice in the center of the cube. It was initially set at a higher value of $\bar{r} = 2$ and then gradually lowered until the results improved. If I already had a fault model or automatic fault detector, such as a coherency cube (Bahorich and Farmer, 1995), I could have passed that to the inversion as a weight instead of implementing the less robust and more computationally expensive IRLS method. The automatically estimated fault weight is shown in solid red in Figure 2.6. The fault weight is slightly shifted from the discontinuity because the weight is identifying poor dips in the flattened cube.

The artifacts within the annotated oval are a result of unflattening a pinchout. Because the reference trace is located in a full (non-pinched out) section, the method will map the same horizon to many locations. The cartoon in Figure 2.4 illustrates this.

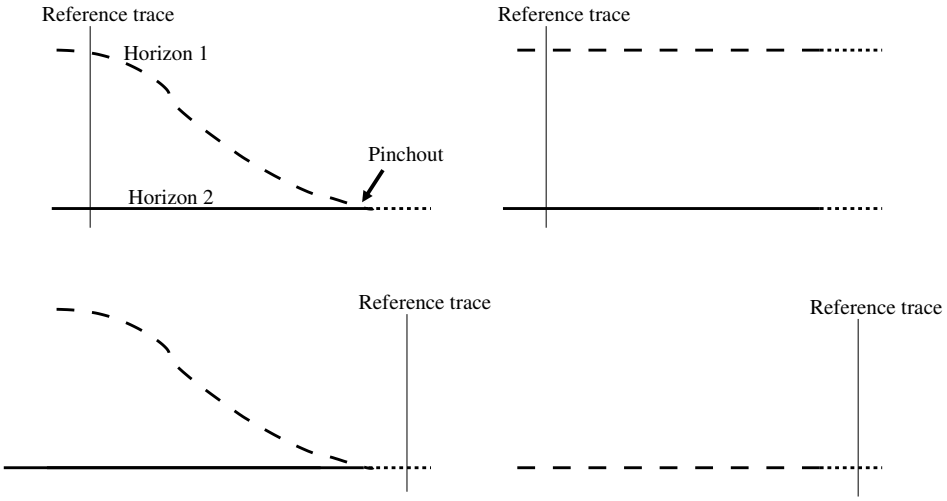

Figure 2.4: A cartoon illustrating the effect of flattening data with a pinchout with different reference trace locations. Top left, two horizons merge forming a pinchout. Top right, after flattening the top left, the dotted portion where the two horizons merged is mapped to two flattened horizons. Bottom left, the reference trace is now located where the horizons have merged. Bottom right, both horizons are mapped to the same location. flat-pinch_cart [NR]

The estimated $\bar{\boldsymbol{\tau}}$ field used to flatten the data is displayed in Figure 2.5 where the reference trace has been substracted as $\bar{\boldsymbol{\tau}}(x,y,t)=\boldsymbol{\tau}(x,y,t)-\boldsymbol{\tau}(x_{\text{ref}},y_{\text{ref}},t)$. The $\bar{\boldsymbol{\tau}}$ field can also be used to reverse the process. That is, I can use the time shifts to unflatten data that is already flat. By unflattening flat surfaces and overlaying them on the data, I am essentially picking any or all of the horizons in the data cube. Figure 2.6 displays every 30th horizon overlaid on the synthetic data shown in Figure 2.2. I could just as easily have displayed every horizon but the image would appear too cluttered. Notice that the horizons are well tracked even across a fault. Additionally, the slight shifts in fault weight can be removed by unflattening the fault weight itself.

Gulf of Mexico Salt Piercement Data

Figure 2.7 is a field 3D data cube from the Gulf of Mexico provided by Chevron. It consists of structurally simple horizons that have been warped up around a salt piercement. Several channels can be seen in the time slice at the top of Figure 2.7 but are largely obscured by the gradual amplitude overprint of a nearly flat horizon that is cut by the time slice. Also, hints of several channels can be seen near the salt piercement.

Figure 2.8 shows the flattened output of the data in Figure 2.7. I flattened this data set using the method without weights. The seismic cube has been converted from a stack of time slices to a stack of horizon slices. Notice that the gradual amplitude overprint present in the unflattened data is no longer present in the flattened data. This is because horizons are no longer cutting across the image. Several channels are now easily visible on the horizon slice. Also, the beds adjacent to the salt dome have been partially reconstructed causing the salt to occupy a smaller area in Figure 2.8. Additionally, a channel (circled) can be seen adjacent to the salt.

The $\bar{\boldsymbol{\tau}}$ field (reference trace subtracted) used to flatten the data has been roughened with the helix derivative (Claerbout, 1999) and is displayed in Figure 2.9. Notice channels are clearly visible. This means that the subtle dip changes from the channel boundaries were picked up by the dip estimation and subsequently used for flattening. This means that the flattening result in Figure 2.8 could possibly reveal more channel information by first applying a low pass filter

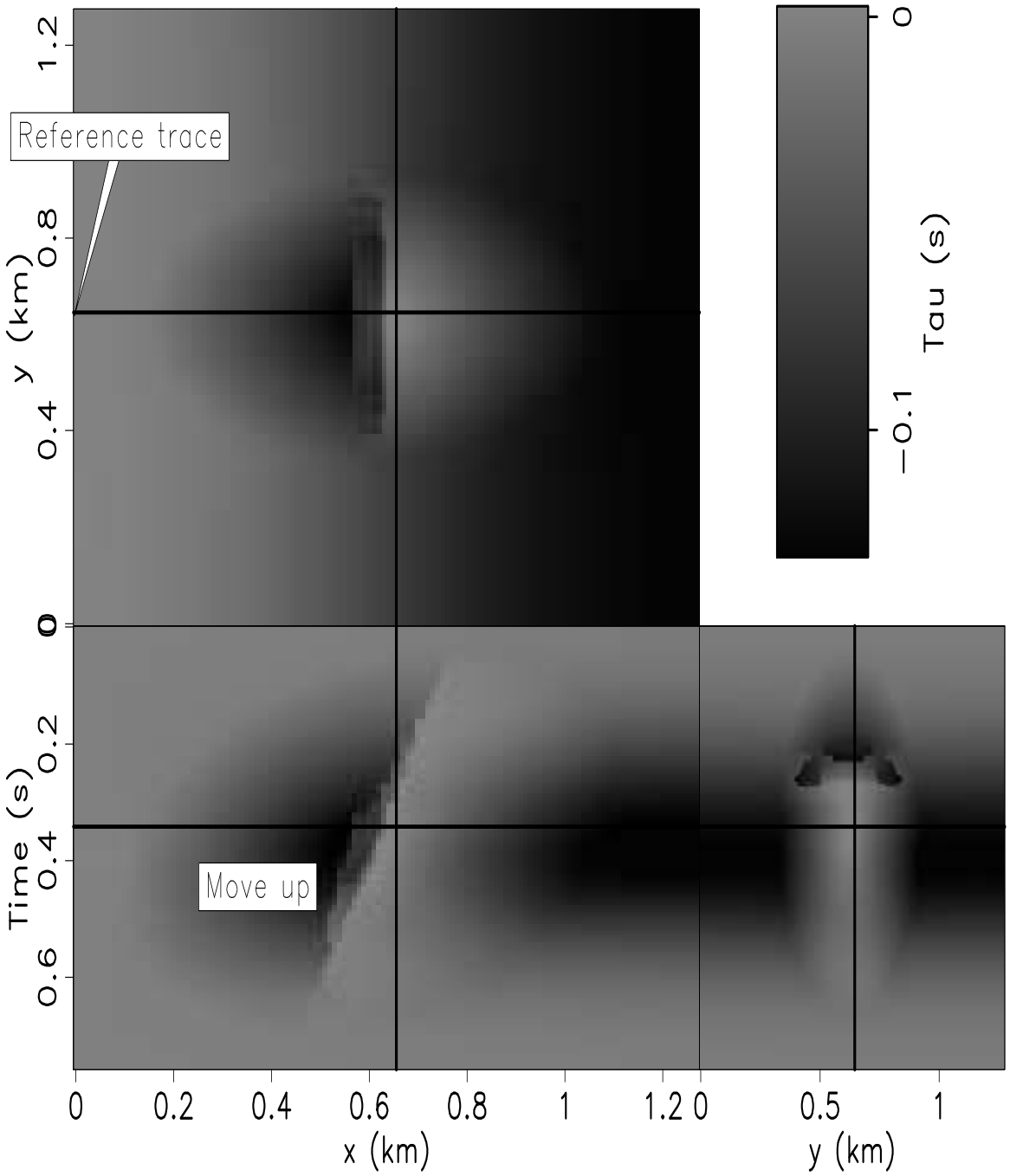

Figure 2.5: The same as Figure 2.2 except showing the $\bar{\tau}$ field used for flattening. The darker grey means the reflectors will be moved up, whereas lighter means moved down. flat-downlap_tau [ER]

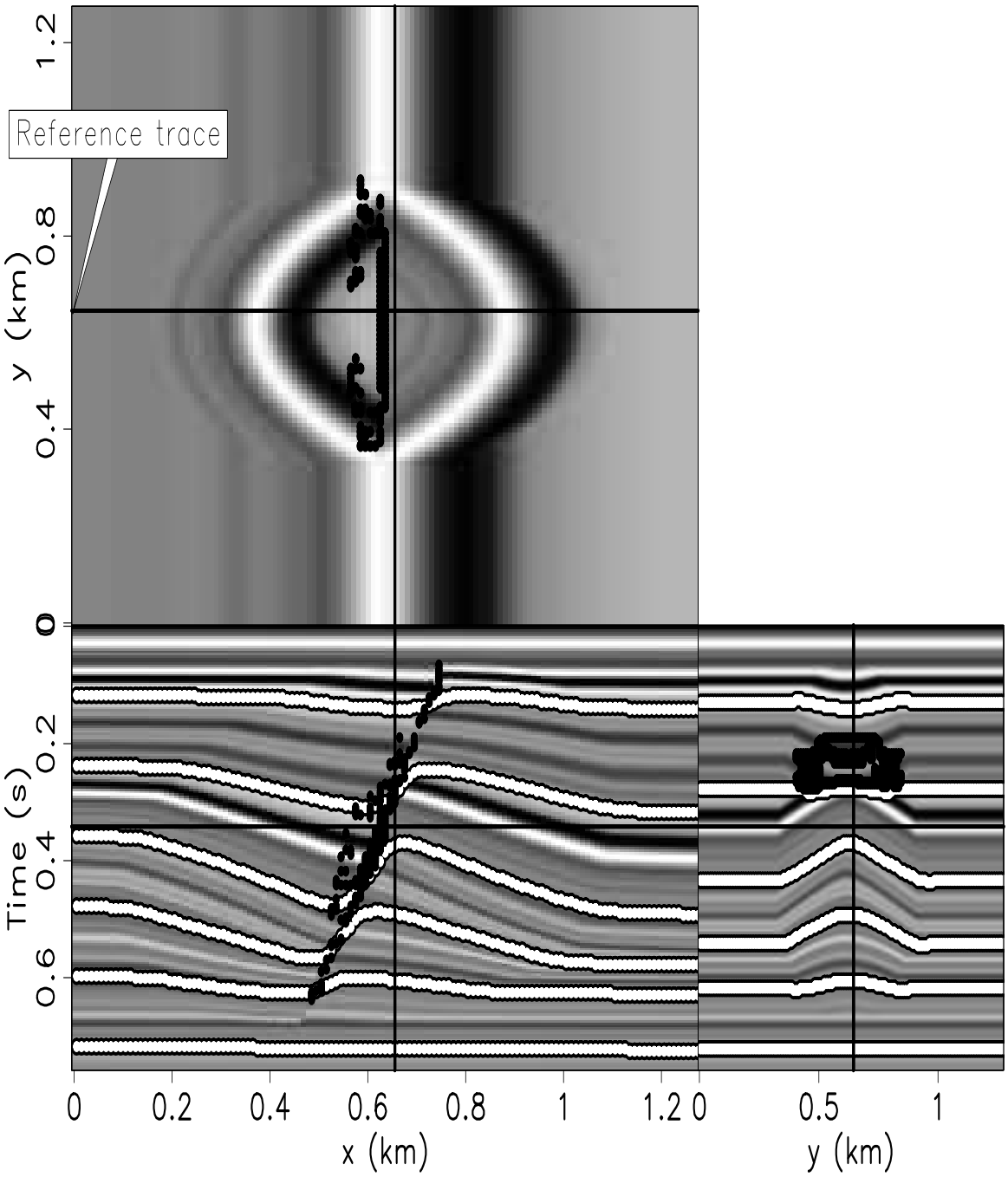

Figure 2.6: Result of overlaying tracked horizons on the image in Figure 2.2. The fault weight that was automatically generated by the IRLS approach is displayed in black. The method successfully tracks the horizons (white and black). flat-downlap_pck [ER]

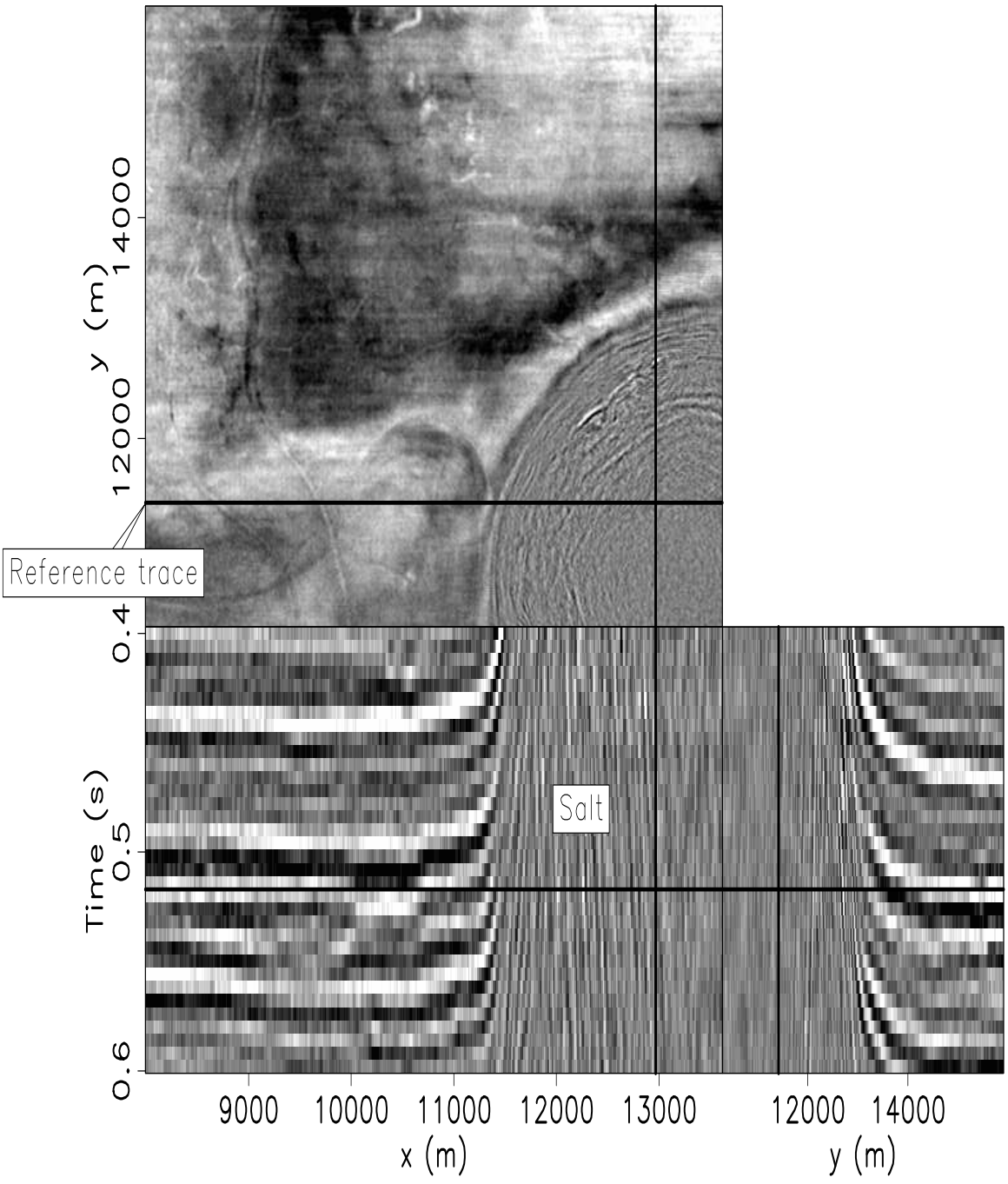

Figure 2.7: Chevron Gulf of Mexico data. The black lines superimposed onto the orthogonal sections identify the location of these sections: a times slice at time=0.522 s, an in-line section at y=11436 m, and a cross-line section at x=12911 m. The reference trace is located at x=8000 m and y=11436 m. Notice how the beds have been forced up to steep angles by the emplacement of a salt piercement. flat-chev [ER]

or smoothing to the input dips. In this way, the flattening algorithm will preserve channel boundaries. Alternatively, the helix derivative of the $\bar{\boldsymbol{\tau}}$ field as a channel map can be improved by less smoothing of the dips. To flatten this data, I used an ϵ=0. The striations present in the vertical sections could be eliminated by using a larger ϵ.

Figure 2.10 displays three horizons overlain on the original data in Figure 2.7. The horizons track the reflectors on the flanks of the salt well. Within the salt, the horizons gradually diverge from their respective reflector events as the estimated dip becomes less accurate. The time slice at the top displays the swath of a tracked horizon.

North Sea Unconformity Data

Figure 2.11 shows a 3D North Sea data set provided by Total. Marked by considerable folding and a sharp angular unconformity, this data presents a real-world flattening challenge. The flattening result shown in Figure 2.12 was made using the method without weights. I used $\epsilon = 0.0$ to make the result as flat as possible. A stretch mute could be applied during the flattenning process to mute areas where no deposition has occured. However, if the data is allowed to stretch, areas of low frequency in the flattened data (Figure 2.12) identify locations of non-deposition (hiatus). This is something like a Wheeler diagram (Wheeler, 1958). On the other hand, to preserve the continuity of the data, I could have used a larger (non-zero) smoothing parameter in equation (2.15). Consequently, the trade-off between continuity and flatness emerges in cases of pinch outs and unconformities.

The $\bar{\boldsymbol{\tau}}$ field used to flatten the data is displayed in Figure 2.13 where the reference trace has been substracted as $\bar{\boldsymbol{\tau}}(x,y,t)$=$\boldsymbol{\tau}(x,y,t)$-$\boldsymbol{\tau}(x_{\text{ref}},y_{\text{ref}},t)$. It it clear that the $\bar{\boldsymbol{\tau}}$ field is sensitive to location of the angular unconformity. The values of the $\bar{\boldsymbol{\tau}}$ field are in depth units.

To achieve good agreement between the tracked horizons and the data, I found an $\epsilon = 0.0$ to be preferable, i.e., no imposed continuity. The result is shown in Figure 2.14. The time slice at the top shows the swaths of a few horizons. Overall, the tracked horizons track up to and along the unconformity, although some significant errors occur where the data quality is poor and, as a result, the estimated dips are inaccurate.

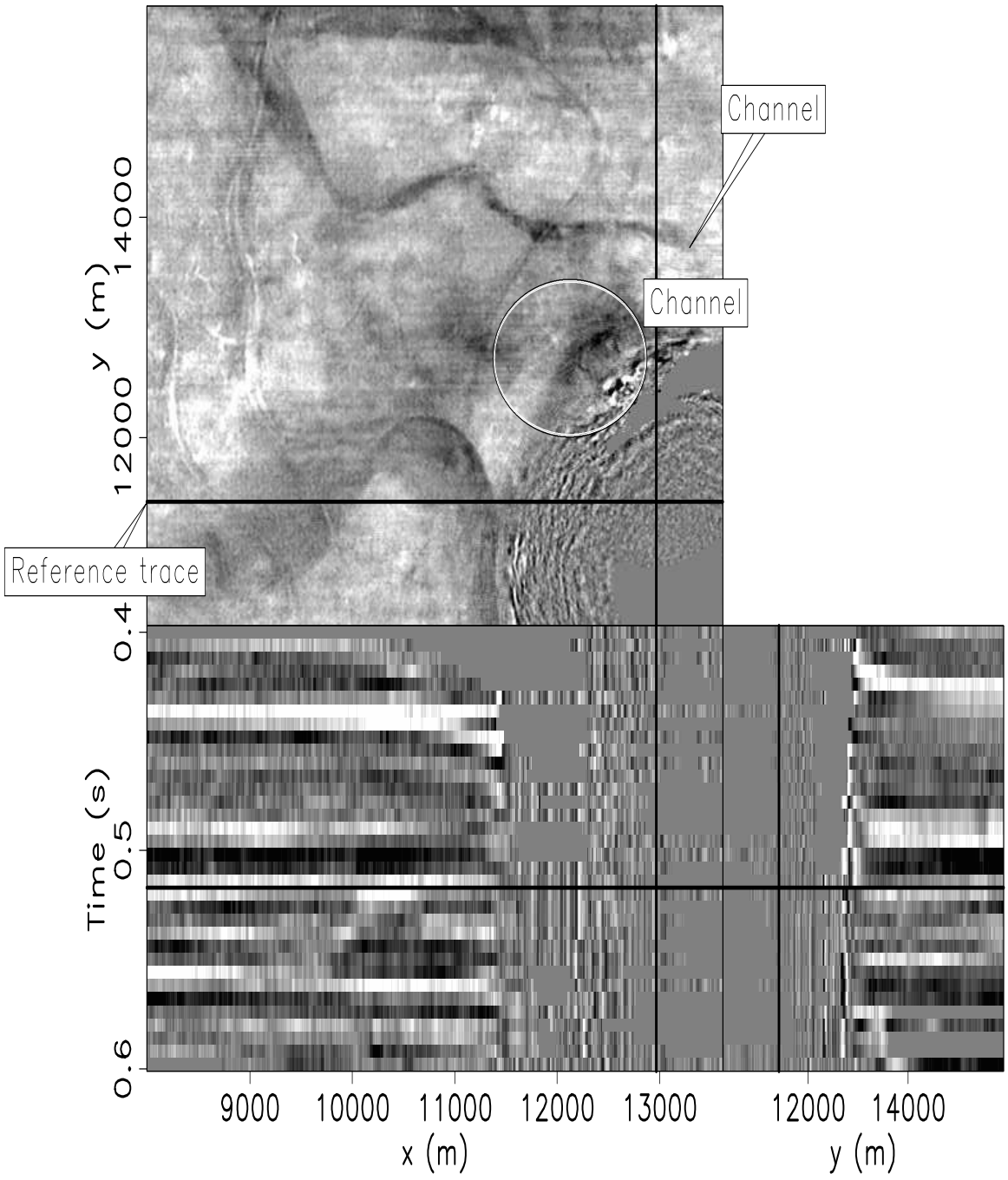

Figure 2.8: As Figure 2.7 only after flattening. The top panel is now a horizon slice displaying several clearly visible channels. flat-chev_flat [ER]

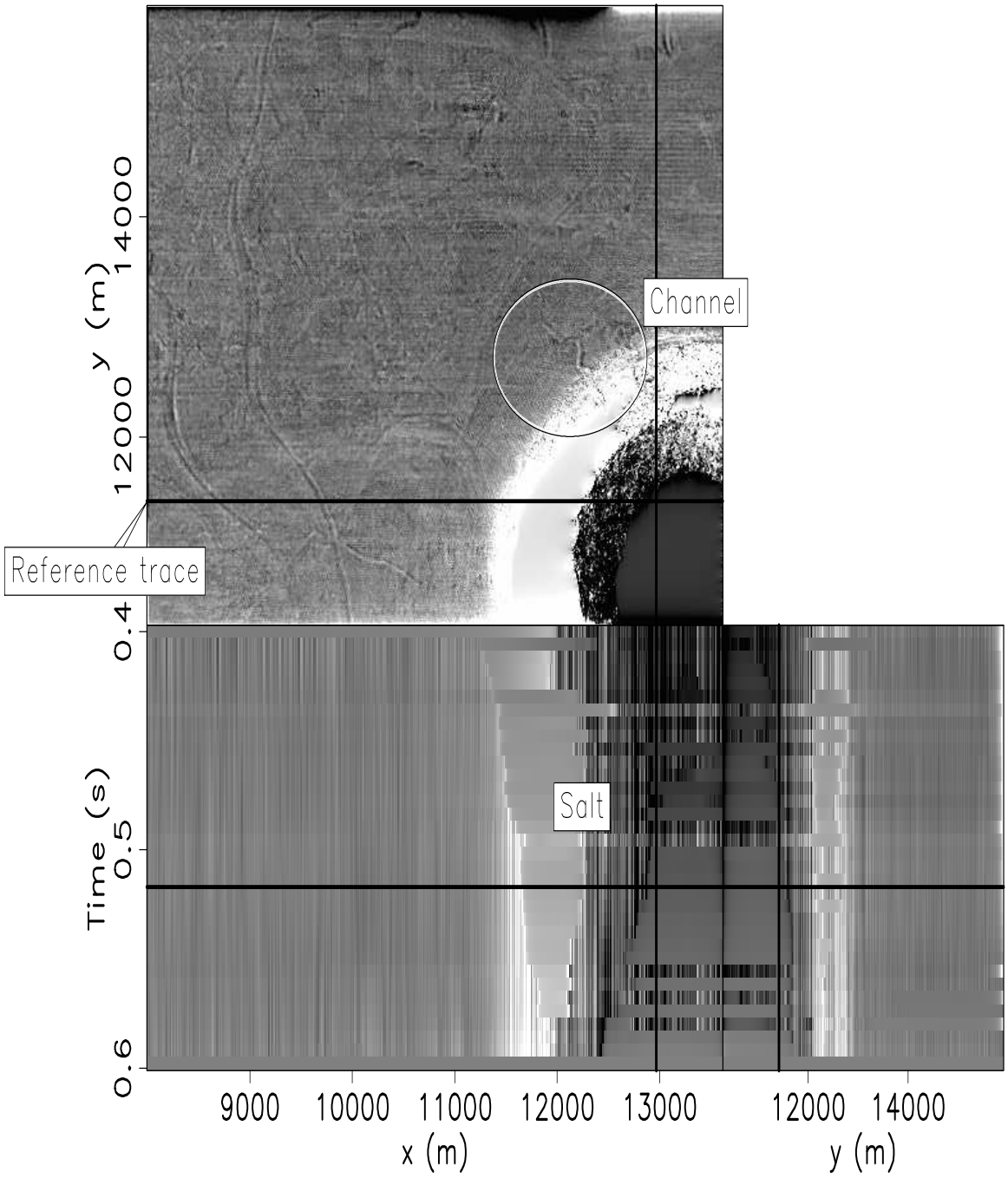

Figure 2.9: As Figure 2.7 showing the $\bar{\tau}$ field used for flattening. To remove low frequency trends, the volume was roughened with the helix derivative in the (x, y)-plane. The shapes of several channels have been captured by the flattening process. flat-chev_tau [ER]

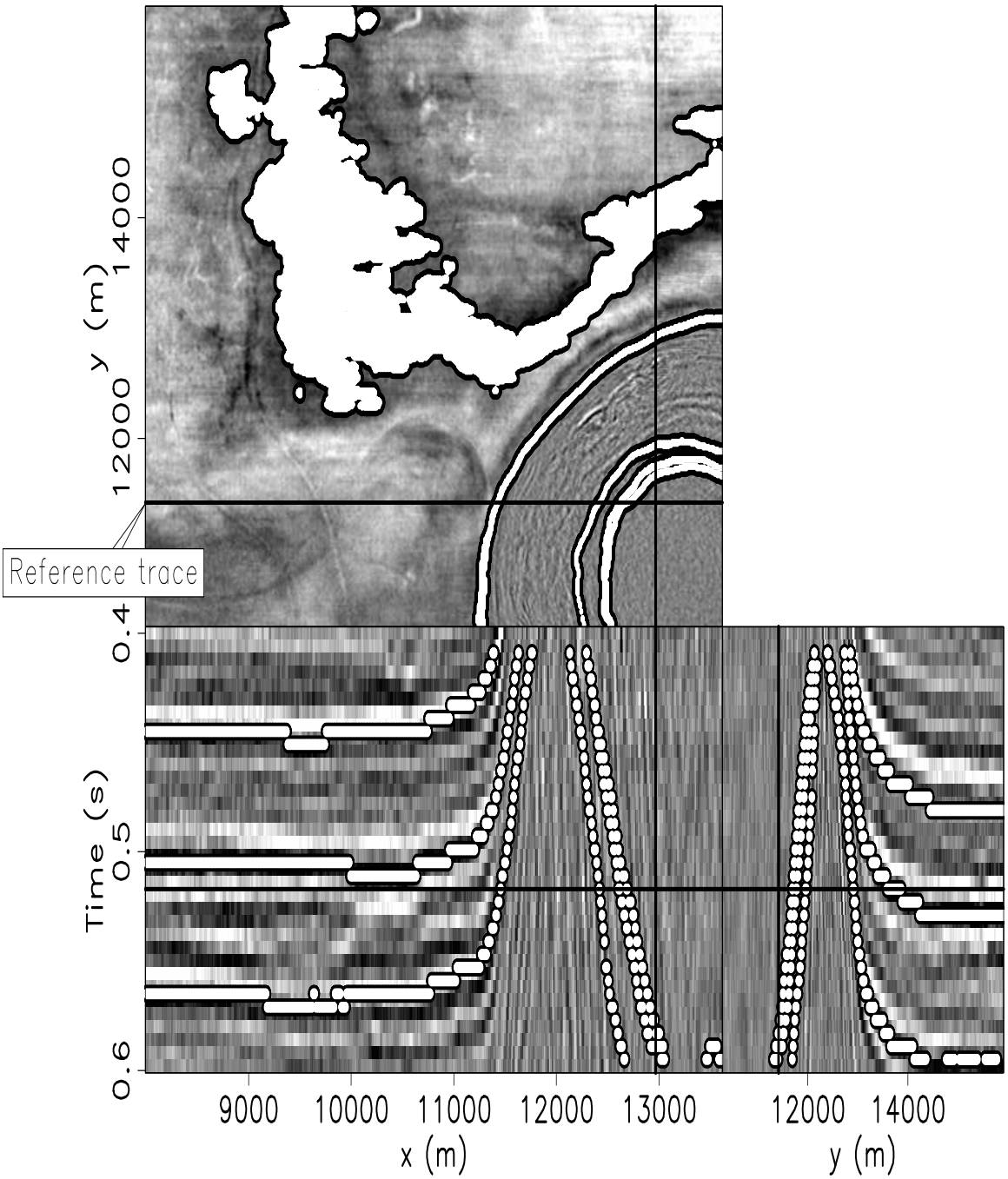

Figure 2.10: Result of overlaying tracked horizons on the image in Figure 2.7. The horizons have been accurately tracked even up to the considerably steep dips leading into the salt piercement. flat-chev_pck [ER]

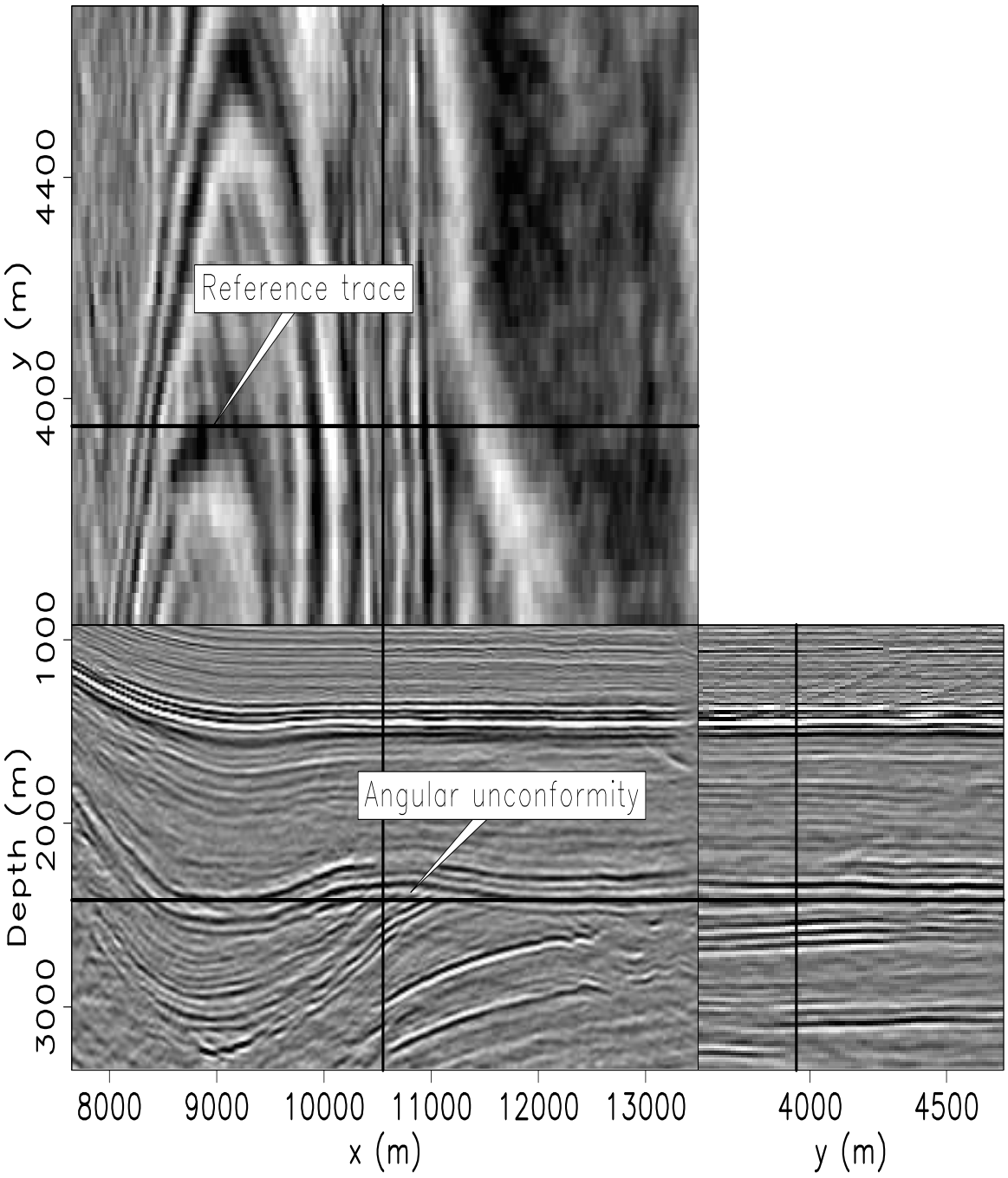

Figure 2.11: North Sea data. The black lines superimposed onto the orthogonal sections identify the location of these sections: a depth slice at depth=2425 m, an in-line section at y=3960 m, and a cross-line section at x=10560 m. The reference trace is located at x=8980 m and y=3960 m. Note the angular unconformity at 2425 meters. flat-elf1 [ER]

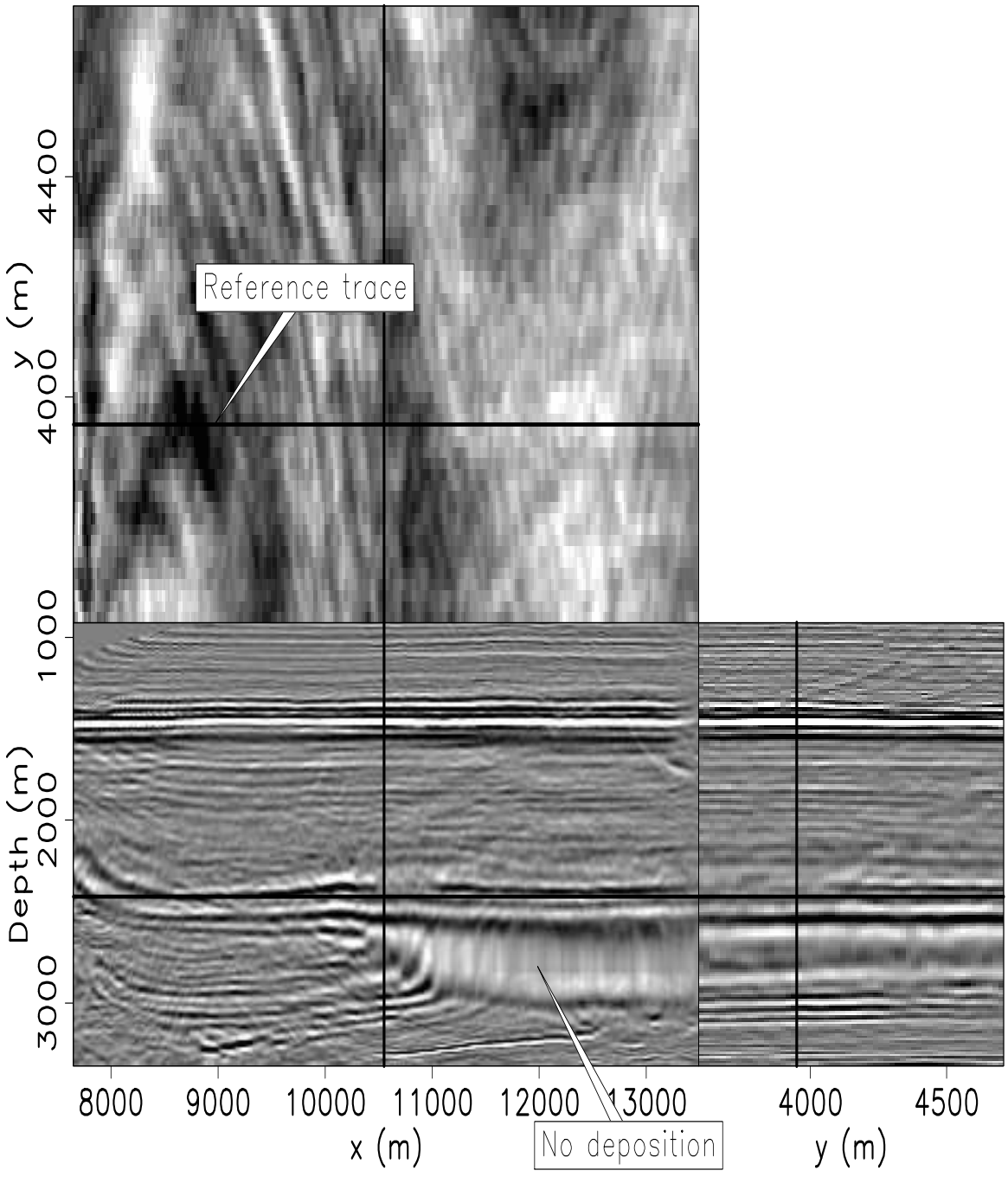

Figure 2.12: As Figure 2.11 only after flattening without weighting. The data is stretched in areas of non-deposition. flat-elf1_flat [ER]

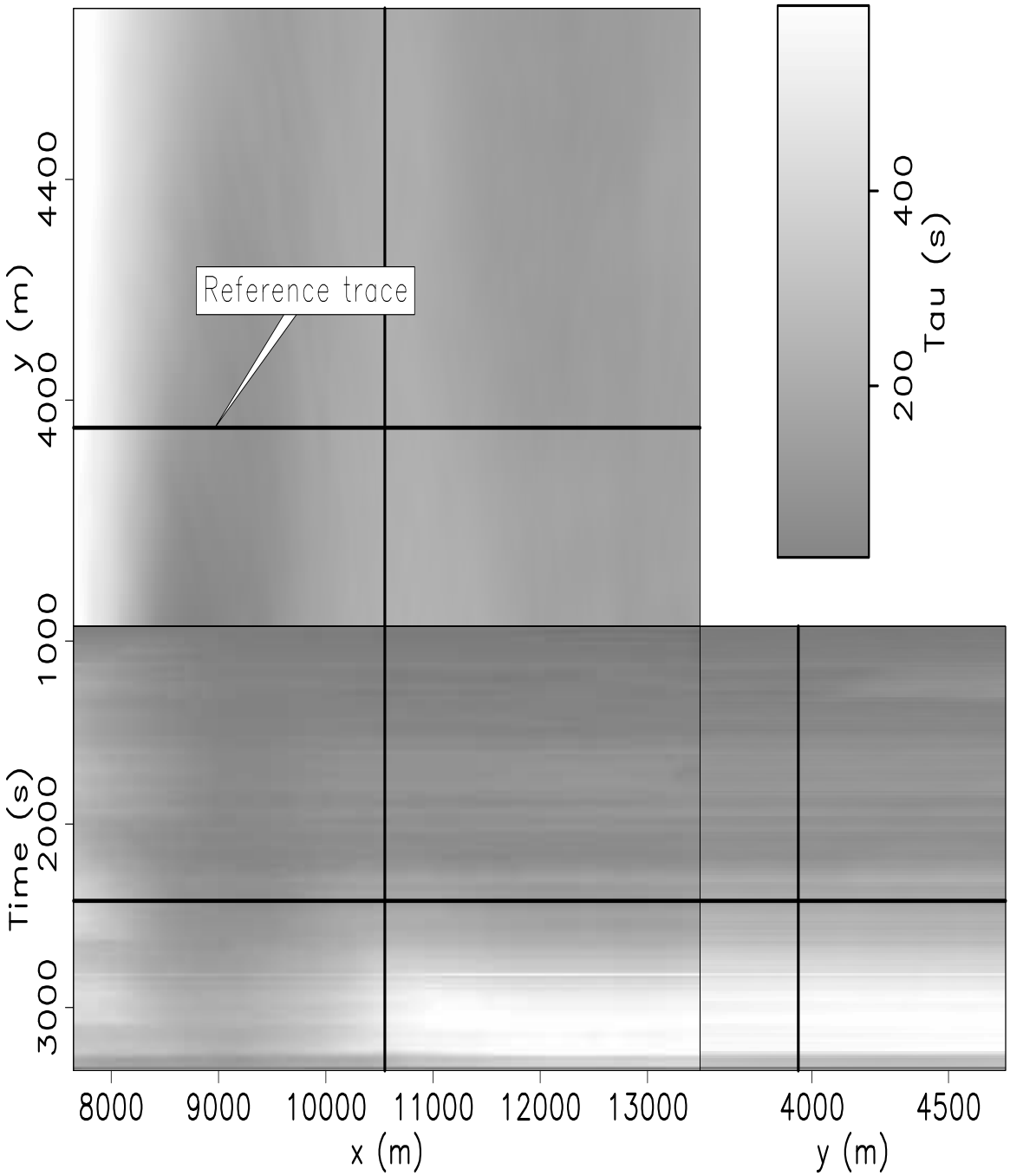

Figure 2.13: As Figure 2.11 showing the $\bar{\tau}$ field used for flattening. The reference trace location is chosen to go through the thickest section in order to preserve as much data as possible in the flattening process. flat-elf1_tau [ER]

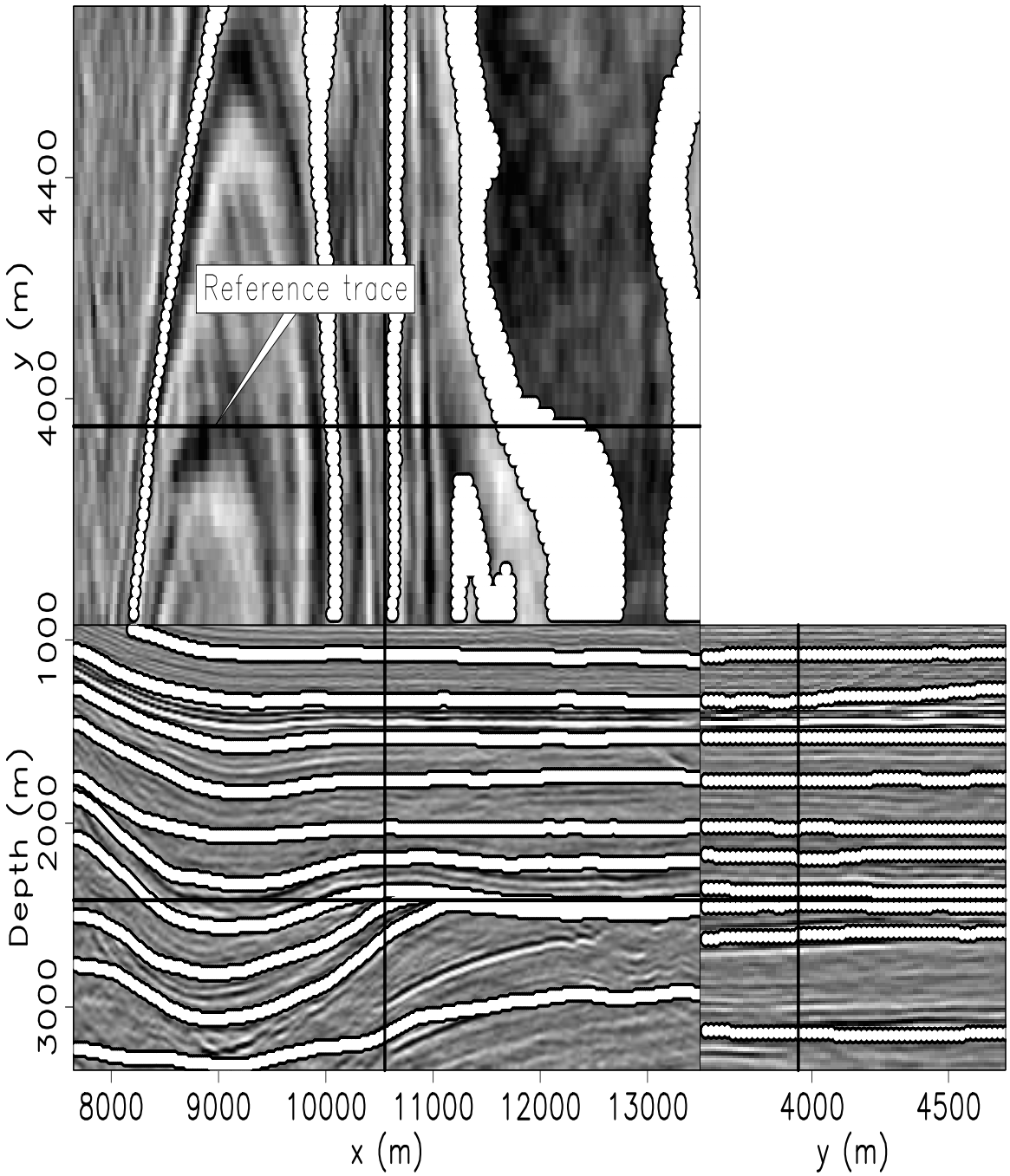

Figure 2.14: The result of overlaying tracked horizons on the image in Figure 2.11. Here I used a smoothing parameter ϵ=0.0 causing horizons that lead to the angular unconformity to be tracked. flat-elf1_pck [ER]

This data set also illustrates the importance of the location of the reference trace (see Figure 2.4). Since our current method requires selection of a single trace (in theory, it could be a different location for each slice), I recommend that the location of the reference trace should be chosen to go through the thickest section in order to preserve as much of the data as possible. It is for this reason that I place the reference located at x=8980 m and y=3960 m in Figures 2.12-2.14. A consequence of choosing the reference trace through the full section will cause the unconformity to be mapped to multiple locations in flattened space (see annotation in Figure 2.12). For comparison, it is interesting to place the reference trace through relatively thin sediments (again, see Figure 2.4). In Figures 2.15-2.17 I chose the reference trace to go through a section where many of the horizons have been pinched out. In the $\bar{\boldsymbol{\tau}}$ field in Figure 2.16, the shifts are now reversed as compared to Figure 2.13; events that were mapped down in depth are now mapped up. Finally many of the events that pinchout in the lower left of the data cube are simply not picked when a poor choice of reference trace position is made. Notice the lack of picked reflections in the lower left of Figure 2.17. These events are overwritten in Figure 2.15.

Gulf of Mexico Faulted Data

Figure 2.18 is an image of a Gulf of Mexico data set containing a fault provided by Chevron. I use the IRLS method to simultaneously flatten this data cube and estimate a fault weight. This IRLS method uses the residual to gradually identify areas to ignore incorrect dips. The damping parameter value $\bar{r} = .2$ was found by trial and error on a particular time slice, and subsequently applied to the entire cube. The flattening results are shown in Figure 2.19. Notice that the horizons are flat, even directly across from the fault. The $\bar{\boldsymbol{\tau}}$ (reference trace removed) field used is displayed in Figure 2.20. Figure 2.21 shows the original data with two unflattened horizons overlying it. It successfully tracks the horizons even across the fault. The automatically generated fault weight is shown in solid black. This IRLS method works best when the dip estimates at the fault vary signicantly from the mean dip of the cube; that the inaccurate fault dips are outliers. For this reason the dips were not smoothed laterally, only vertically. It is also necessary for this reason to choose a dip estimator that is not well behaved at faults.

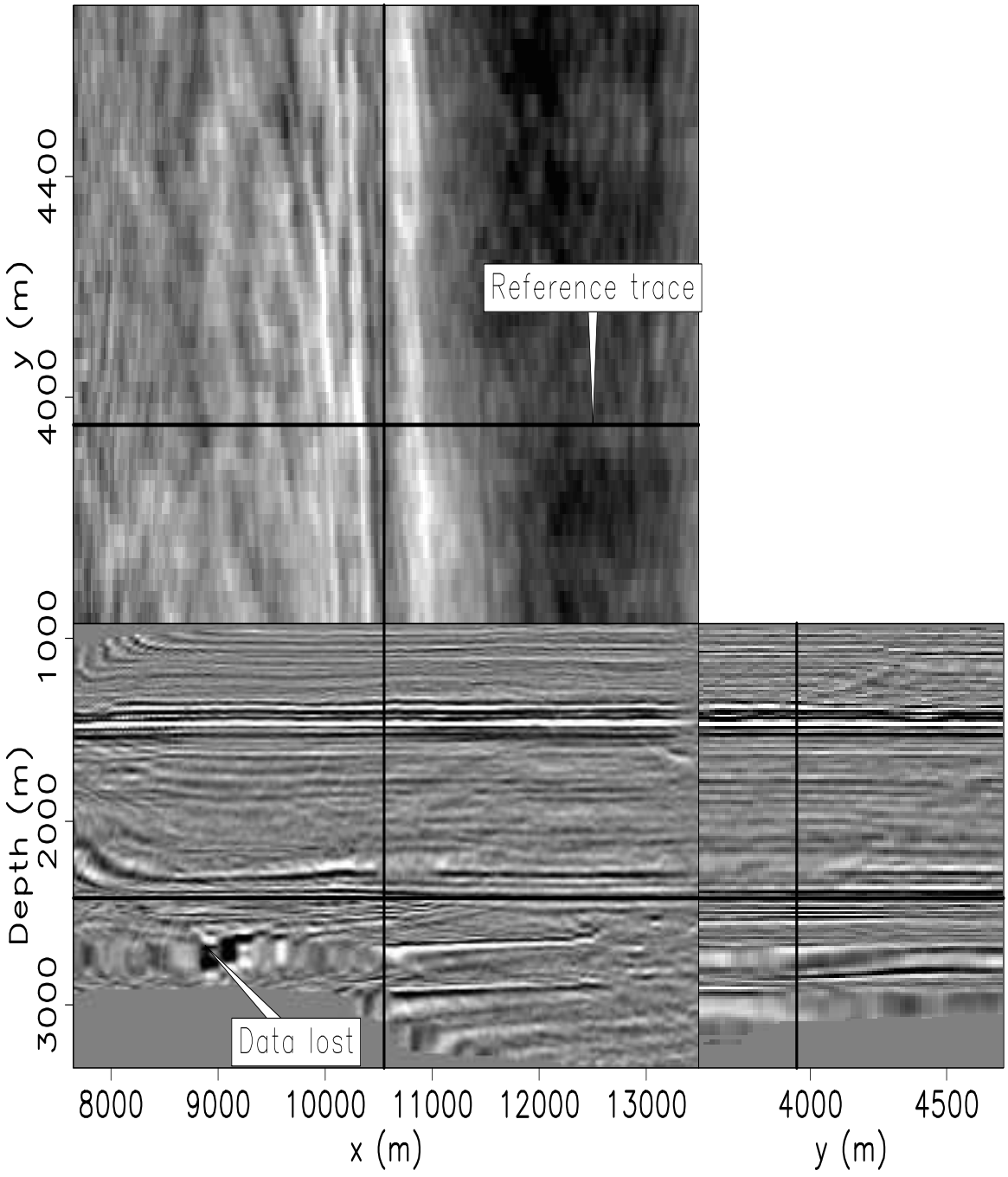

Figure 2.15: The same as Figure 2.12 except the reference trace is located above an area of non-deposition at x=12500 m and y=3960 m. Several of the horizons in the lower left corner were mapped on top of eachother (data was lost). Coincidentally, the overlying horizons were stretched. flat-elf_thin_flat [ER]

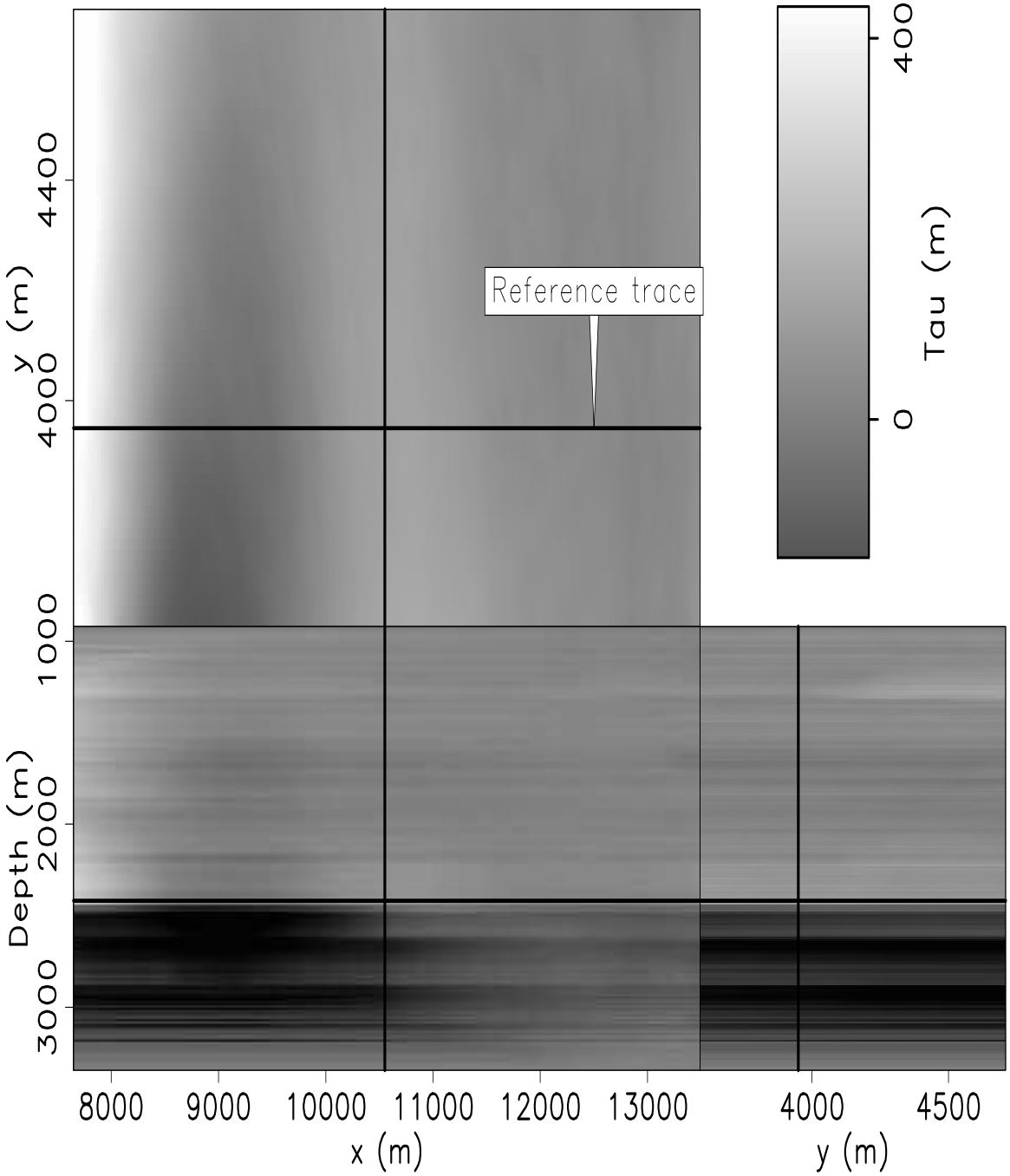

Figure 2.16: As Figure 2.15 showing the $\bar{\tau}$ field used for flattening. Notice the effect of the position of the reference trace on the $\bar{\tau}$ field by comparing to Figure 2.13. flat-elf_thin_tau [ER]

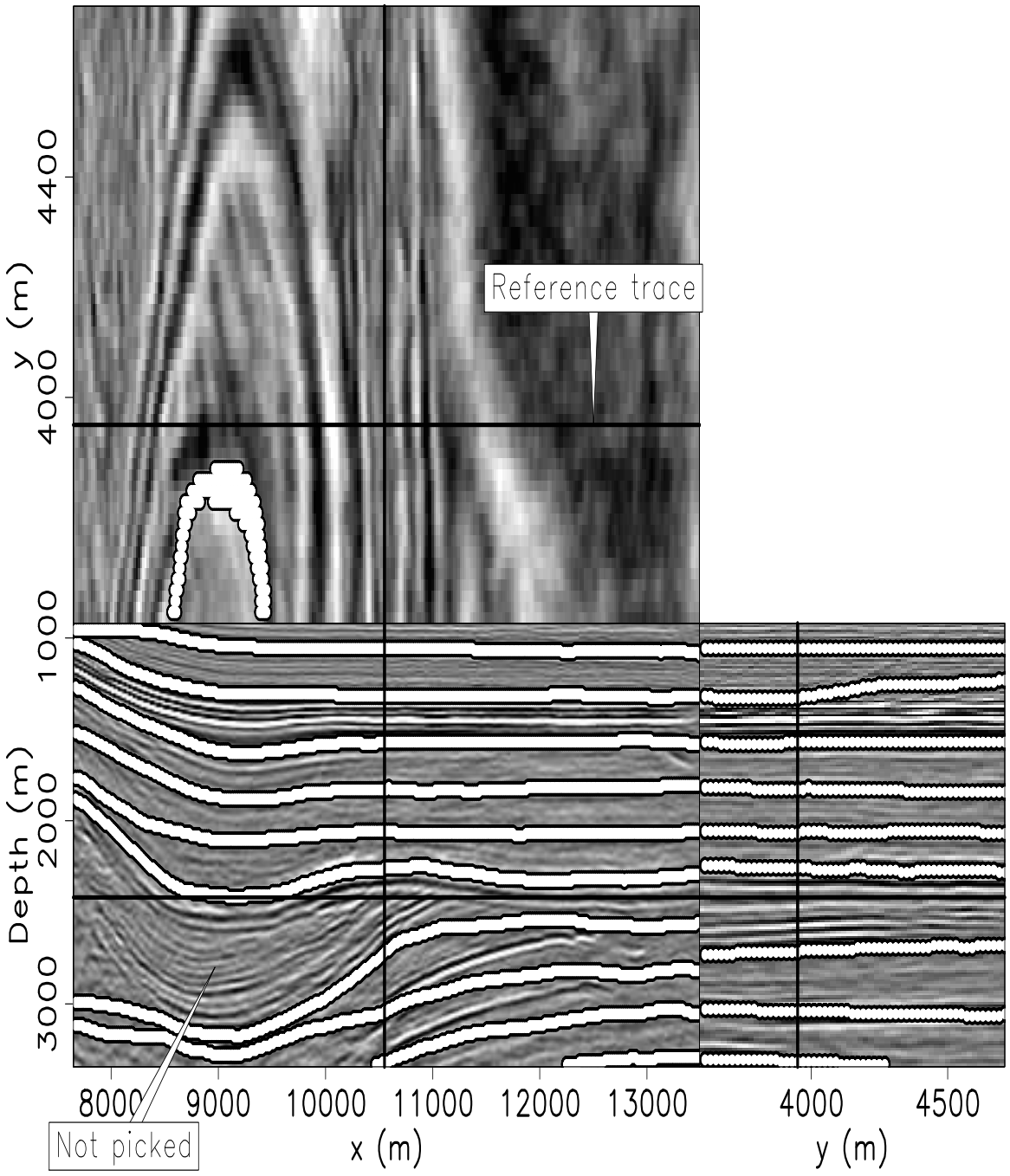

Figure 2.17: The result of overlaying tracked horizons on the image in Figure 2.11 with a different reference trace. Notice that horizons are not tracked at all in the annotated region. flat-elf_thin_pck [ER]

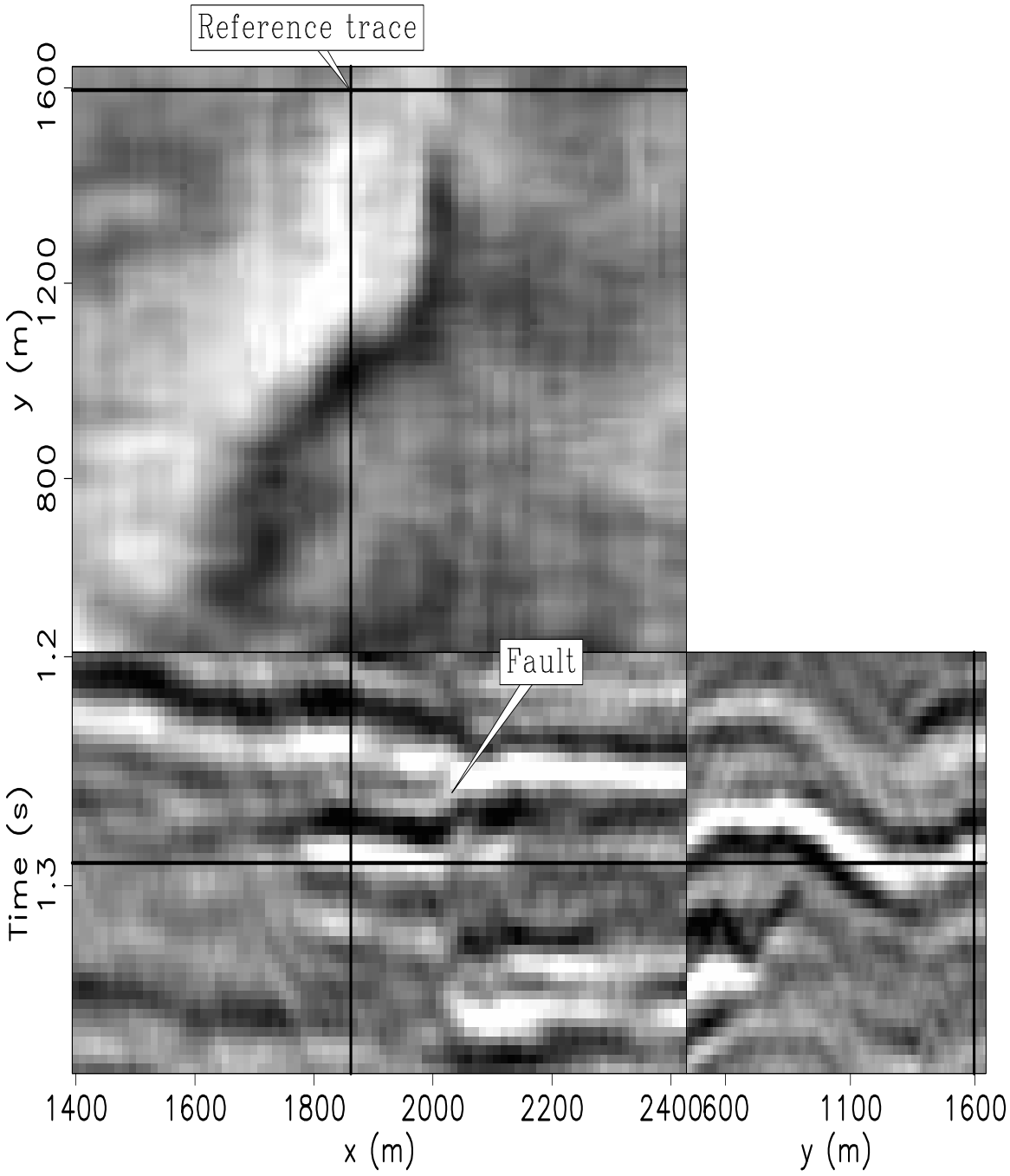

Figure 2.18: Gulf of Mexico data set displaying a fault. The black lines superimposed onto the orthogonal sections identify the location of these sections: a time slice at time=1.292 s, an in-line section at y=1602 m, and a cross-line section at x=1868 m. The reference trace is located at x=1868 m and y=1602 m. flat-shoal [CR]

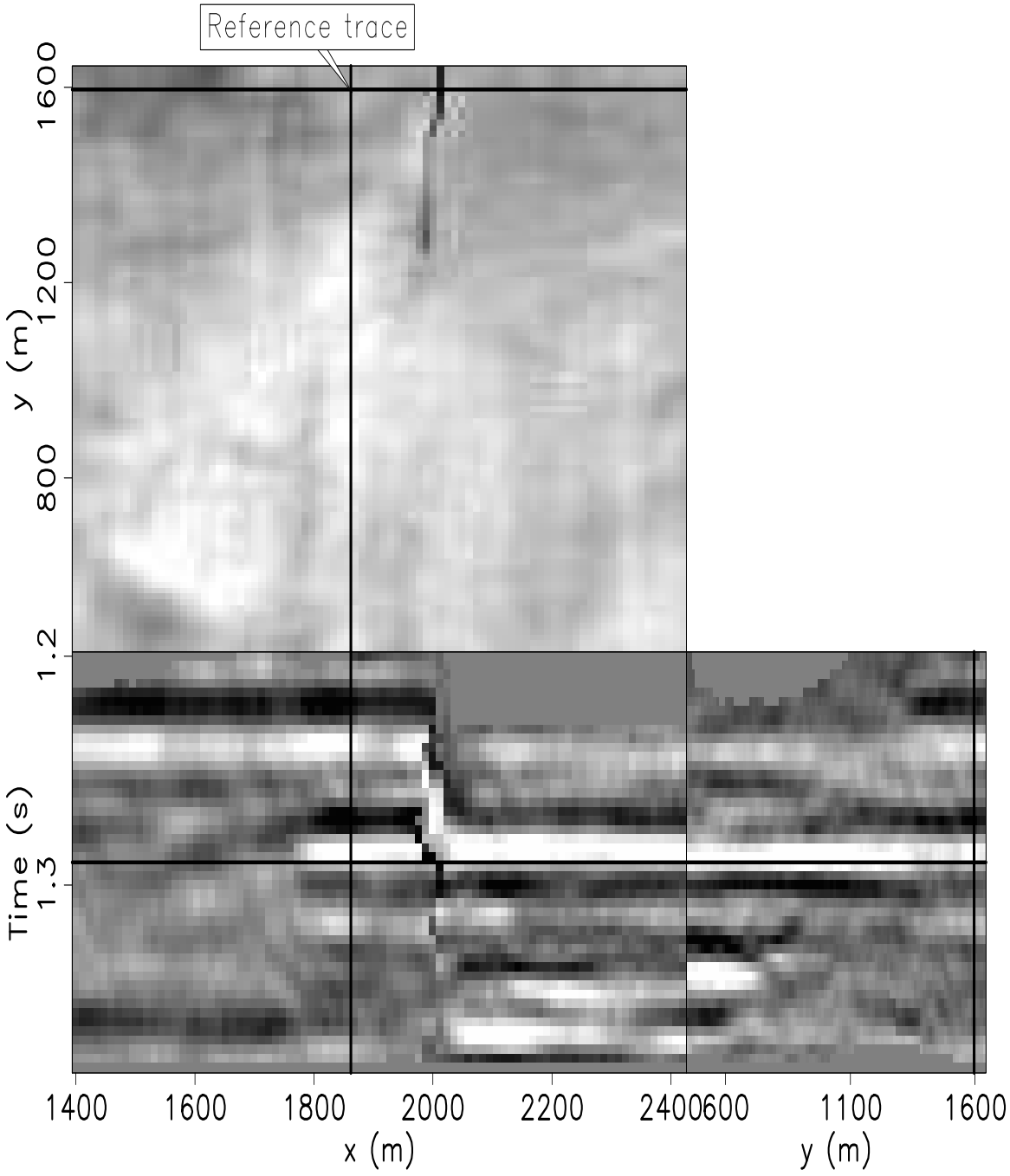

Figure 2.19: As Figure 2.18 only after flattening using IRLS method. flat-shoal_flat [CR]

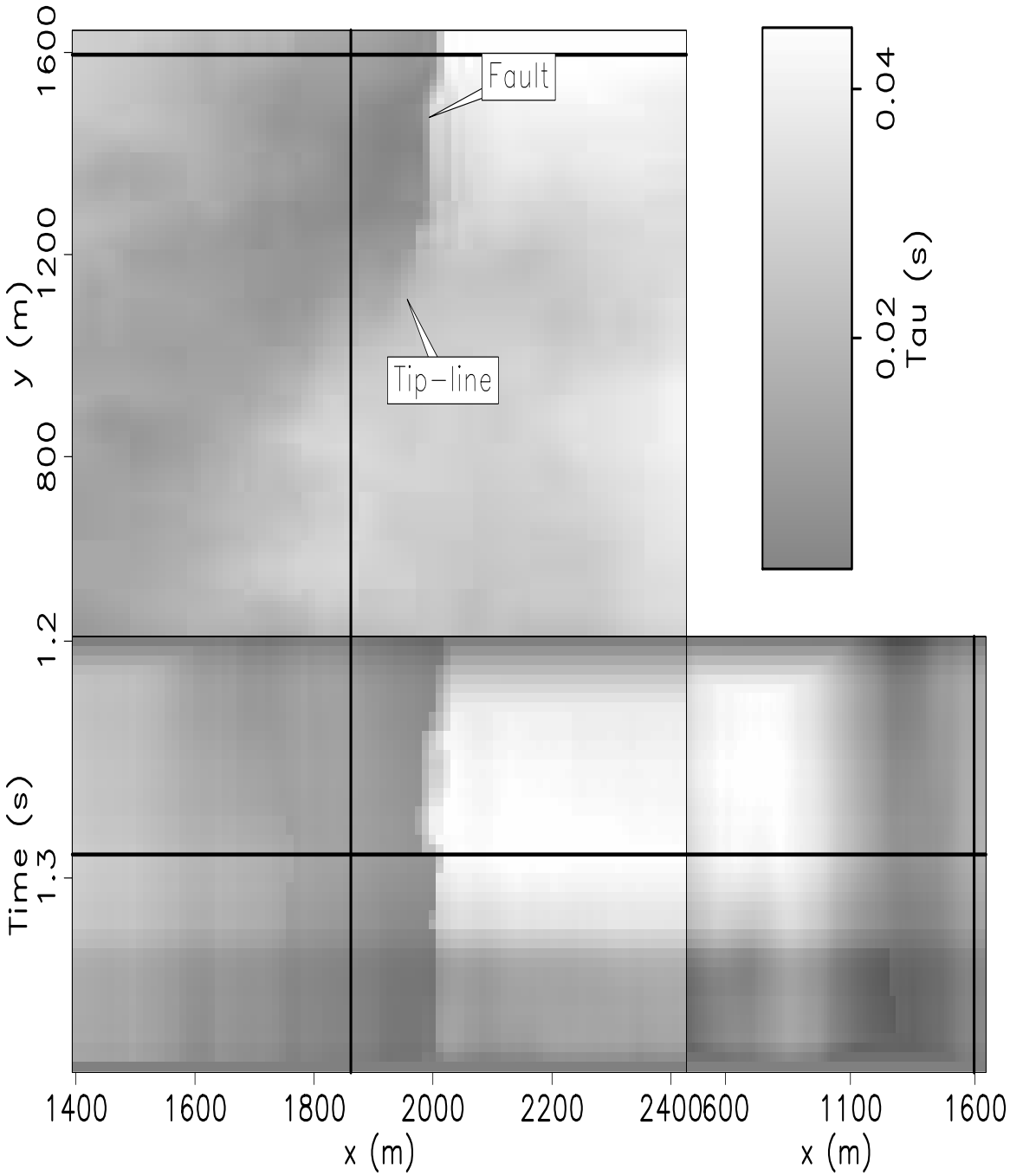

Figure 2.20: As Figure 2.18 showing the $\bar{\tau}$ field used for flattening. Notice that sudden changes in the $\bar{\tau}$ field occur at the fault. Also, the fault terminates within the cube (tip-line). flat-shoal_tau [CR]

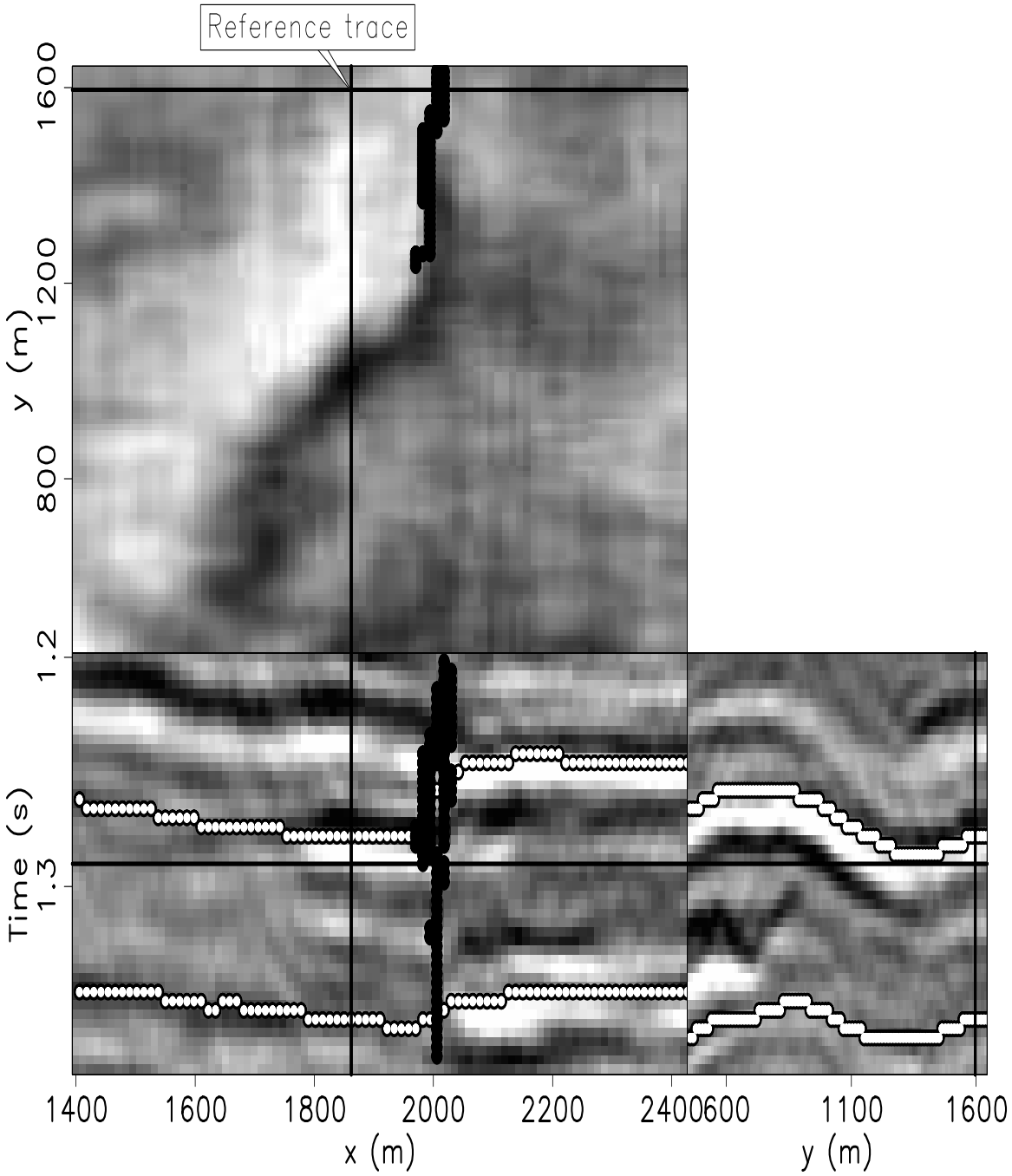

Figure 2.21: The result of overlaying two tracked horizons and the automatically generated fault weight (black) on the image in Figure 2.18. flat-shoal_pck [CR]

FLATTENING WITH GEOLOGICAL CONSTRAINTS

A most attractive feature of the previously described flattening technique is that it requires no manual picking. This would be fine if all data sets had reasonably accurate estimated dips, but in the real world automatic flattening can produce results that are not perfect. Noise, both coherent and otherwise, can overwhelm the dip estimation causing reflectors in those areas to not be flat. Multiples, for example, are significant source of coherent noise that can contaminate the dip field. Although certain faults with tip-lines (terminations) encased within the data cube can be flattened if a fault model is provided, faults that cut across the entire data cube cannot. Consequently, it would be useful to have the ability to add some geological constraints to restrict the flattening result in areas of poor data quality while allowing it to efficiently tackle other areas where estimated dips are more accurate. Additionally, correlations can be used as constraints to reconstruct across faults that do not terminate within the data cube.

In this section, I present flattening with hard constraints. This was initially presented by Lomask and Guitton (2006a,b). The hard constraints can be manually picked horizons or individual picks. Regularization is required to spread information from the constraints throughout the data cube. The hard constraints are implemented as a masked initial model within the inversion. I envision a tool that interpreters can run once completely unconstrained, then quality control the results. The interpreter can then adjust some horizons and then run the flattening method again honoring their changes. The algorithm is fast enough so that this process can be repeated several times. In the future, computational and algorithmic improvements can result in a flattening method that is so efficient that the flattening process can be run between picks. Further, this method has the potential of combining other information into the flattening such as well log picks.

Adding constraints

Recall that the residual in equation (2.15) for the 3D method is defined as

$$\mathbf{r} \quad = \quad \nabla_\epsilon \boldsymbol{\tau} - \mathbf{p}_\epsilon \quad = \quad \begin{bmatrix} \frac{\partial \boldsymbol{\tau}}{\partial x} \\ \frac{\partial \boldsymbol{\tau}}{\partial y} \\ \epsilon \frac{\partial \boldsymbol{\tau}}{\partial t} \end{bmatrix} \quad - \quad \begin{bmatrix} \mathbf{b} \\ \mathbf{q} \\ \mathbf{0} \end{bmatrix}. \tag{2.30}$$

I wish to add a model mask $\mathbf{K}$ to prevent changes to specific areas of an initial $\boldsymbol{\tau}_0$ field. This initial $\boldsymbol{\tau}_0$ field can be picks from any source. In general, they may come from a manually picked horizon or group of horizons. These initial constraints do not have to be continuous surfaces but instead can be isolated picks, such as well-to-seismic ties.

It is important that the initial $\boldsymbol{\tau}_0$ field is zero at the reference trace. This means that for every isolated pick, the value of the pick is the time shift from that point to where the event crosses the reference trace. In other words, the initial $\boldsymbol{\tau}_0$ is defined as $\boldsymbol{\tau}_0 = \boldsymbol{\tau}(x,y,t)$-$\boldsymbol{\tau}(x_{\text{ref}}, y_{\text{ref}}, t)$

To apply the mask I follow the same the development as Claerbout (1999) as

$$\begin{aligned}
\mathbf{0} &\approx \nabla_\epsilon \boldsymbol{\tau} - \mathbf{p}_\epsilon && (2.31)\\
\mathbf{0} &\approx \nabla_\epsilon (\mathbf{K} + (\mathbf{I} - \mathbf{K}))\boldsymbol{\tau} - \mathbf{p}_\epsilon && (2.32)\\
\mathbf{0} &\approx \nabla_\epsilon \mathbf{K}\boldsymbol{\tau} + \nabla_\epsilon (\mathbf{I} - \mathbf{K})\boldsymbol{\tau} - \mathbf{p}_\epsilon && (2.33)\\
\mathbf{0} &\approx \nabla_\epsilon \mathbf{K}\boldsymbol{\tau} + \nabla_\epsilon \boldsymbol{\tau}_0 - \mathbf{p}_\epsilon && (2.34)\\
\mathbf{0} \quad \approx \quad \mathbf{r} &= \nabla_\epsilon \mathbf{K}\boldsymbol{\tau} + \mathbf{r}_0 - \mathbf{p}_\epsilon. && (2.35)
\end{aligned}$$

The resulting equations are now

iterate {

$$\begin{aligned}
\mathbf{r} &= \nabla_\epsilon \mathbf{K}\boldsymbol{\tau}_k - \mathbf{p}_\epsilon(\boldsymbol{\tau}_k) + \mathbf{r}_0 && (2.36)\\
\Delta\boldsymbol{\tau} &= (\mathbf{K}^{\mathrm{T}} \nabla_\epsilon^{\mathrm{T}} \nabla_\epsilon \mathbf{K})^{-1} \mathbf{K}^{\mathrm{T}} \nabla_\epsilon^{\mathrm{T}} \mathbf{r} && (2.37)
\end{aligned}$$

$$\boldsymbol{\tau}_{k+1} \quad = \quad \boldsymbol{\tau}_k + \Delta\boldsymbol{\tau} \tag{2.38}$$

} .

Because the $\mathbf{K}$ in equation (2.37) is non-stationary I cannot solve it with equation (2.19). Consequently, I solve it in the time domain using preconditioned conjugate gradients. The preconditioner is $(\boldsymbol{\nabla}_\epsilon^{\mathrm{T}}\boldsymbol{\nabla}_\epsilon)^{-1}$ which I solve in the Fourier domain in a manner similar to equation (2.19).

In order to use preconditioned conjugate gradients as defined by (Strang, 1986), I need to define the forward operator so that its inverse is approximated by the preconditioner $(\boldsymbol{\nabla}_\epsilon^{\mathrm{T}}\boldsymbol{\nabla}_\epsilon)^{-1}$. Therefore, following (Ghiglia and Romero, 1994), I define $\mathbf{Q} = \mathbf{K}^{\mathrm{T}}\boldsymbol{\nabla}_\epsilon^{\mathrm{T}}\boldsymbol{\nabla}_\epsilon\mathbf{K}$ and $\bar{\mathbf{r}} = \mathbf{K}^{\mathrm{T}}\boldsymbol{\nabla}_\epsilon^{\mathrm{T}}\mathbf{r}$ so that equation (2.37) becomes

$$\Delta\boldsymbol{\tau} \quad = \quad \mathbf{Q}^{-1}\bar{\mathbf{r}}. \tag{2.39}$$

The computation template for the method of preconditioned conjugate gradients that is solved for each iteration for equation (2.37) is

iterate {
  solve $\mathbf{z}_k \quad = \quad (\boldsymbol{\nabla}_\epsilon^{\mathrm{T}}\boldsymbol{\nabla}_\epsilon)^{-1}\bar{\mathbf{r}}_k$ using equation (2.19)
  subtract off reference trace $\mathbf{z}_k \quad = \quad \mathbf{z}_k - \mathbf{z}_k(ref)$
  if first iteration {
    $\mathbf{p}_1 \quad \longleftarrow \quad \mathbf{z}_0$
  } else{
    $\beta_{k+1} \quad \longleftarrow \quad \bar{\mathbf{r}}_k^{\mathrm{T}}\mathbf{z}_k / \bar{\mathbf{r}}_{k-1}^{\mathrm{T}}\mathbf{z}_{k-1}$
    $\mathbf{p}_{k+1} \quad \longleftarrow \quad \mathbf{z}_k + \beta_{k+1}\mathbf{p}_k$
    }
  $\alpha_{k+1} \quad \longleftarrow \quad \bar{\mathbf{r}}_k^{\mathrm{T}}\mathbf{z}_k / \mathbf{p}_{k+1}^{\mathrm{T}}\mathbf{Q}\mathbf{p}_{k+1}$
  $\Delta\boldsymbol{\tau}_{k+1} \quad \longleftarrow \quad \Delta\boldsymbol{\tau}_k + \alpha_{k+1}\mathbf{p}_{k+1}$
  $\bar{\mathbf{r}}_{k+1} \quad \longleftarrow \quad \bar{\mathbf{r}}_k - \alpha_{k+1}\mathbf{Q}\mathbf{p}_{k+1}$
  }

Here, k is the iteration number starting at k=*0*. It is necessary to subtract off the reference

trace at each iteration because the Fourier based solution using equation (2.19) has no non-stationary information such as the location of the reference trace. Subtracting the reference trace removes a zero frequency shift. Nonetheless, in practice equation (2.19) makes an adequate preconditioner because $\nabla_\epsilon^{\mathrm{T}} \nabla_\epsilon$ is often close to $\mathbf{K}^{\mathrm{T}} \nabla_\epsilon^{\mathrm{T}} \nabla_\epsilon \mathbf{K}$.

Convergence can be determined using the same criterion described in equation (2.12). Alternatively, an adequate stopping criterion would be to consider only the norm of the residual $\|\bar{\mathbf{r}}_k\|$. This is essentially the average of the divergence of the remaining dips, i.e. ,

$$\frac{\|\bar{\mathbf{r}}_k\|}{n} \quad < \quad \mu. \tag{2.40}$$

where $n = n_1 \times n_2 \times n_3$ is the size of the data cube.

Rather than using the cosine transform method (equation (2.19)) as the preconditioner, the helix transform can also be used with the Gauss-Newton approach to approximate the inverse of $\nabla_\epsilon^{\mathrm{T}} \nabla_\epsilon$. This is describe in Appendix B. This method converges to the same results as the above method but is not as fast.

Another method for applying geological constraints is described in the Appendix C. This uses a quasi-Newton technique to solve full equation developed in Appendix A. Because it includes a third term, it has a more accurate gradient direction than the Gauss-Newton methods and, as a result, should require fewer iterations. Unfortunately, each iteration is slower and, as a result, the computation time to converge is greater. This was originally implemented by (Guitton et al., 2005b). This method has the advantage of implementing the constraints within bounds; upper and lower limits of the τ field are passed to the algorithm.

For the three methods, two Gauss-Newton and one quasi-Newton, the memory usage is similar. However, the Gauss-Newton method preconditioned with the cosine transform appears to converge the fastest. The quasi-Newton is the second fastest while the Gauss-Newton method preconditioned with the helix transform is the slowest. However, these results are only preliminary. It seems plausible that the method with the greatest potential would be to solve the full equation in Appendix A with the cosine transform method as the preconditioner.

Constrained solution with weights

In dealing with noise and certain geological features such as faults and angular unconformities, it is necessary to apply a weight to the method. This weight is applied to the residual to ignore fitting equations that are affected by the bad dips estimated at faults. In the case of angular unconformities, it can be used to disable the vertical regularization in locations where multiple horizons converge. The resulting weighted and constrained Gauss-Newton equations are now

iterate {

$$\begin{aligned} \mathbf{r} &= \mathbf{W}\nabla_\epsilon\mathbf{K}\boldsymbol{\tau}_k - \mathbf{W}\mathbf{p}_\epsilon(\boldsymbol{\tau}_k) + \mathbf{W}\mathbf{r}_0 && (2.41)\\ \Delta\boldsymbol{\tau} &= (\mathbf{K}^{\mathrm{T}}\nabla_\epsilon^{\mathrm{T}}\mathbf{W}_\epsilon^{\mathrm{T}}\mathbf{W}\nabla_\epsilon\mathbf{K})^{-1}\mathbf{K}^{\mathrm{T}}\nabla_\epsilon^{\mathrm{T}}\mathbf{W}_\epsilon^{\mathrm{T}}\mathbf{r} && (2.42)\\ \boldsymbol{\tau}_{k+1} &= \boldsymbol{\tau}_k + \Delta\boldsymbol{\tau} && (2.43) \end{aligned}$$

} .

Again, I solve equation (2.42) efficiently using preconditioned conjugate gradients with equation (2.19) as the preconditioner and $\mathbf{Q} = \mathbf{K}^{\mathrm{T}}\nabla_\epsilon^{\mathrm{T}}\mathbf{W}_\epsilon^{\mathrm{T}}\mathbf{W}\nabla_\epsilon\mathbf{K}$ and $\bar{\mathbf{r}} = \mathbf{K}^{\mathrm{T}}\nabla_\epsilon^{\mathrm{T}}\mathbf{W}_\epsilon^{\mathrm{T}}\mathbf{r}$.

CONSTRAINED RESULTS

I conducted tests of the constrained flattening method on two 3D data sets from the Gulf of Mexico and the North Sea. The first example illustrates how a single pair of traces can be manually correlated and passed to the constrained flattening method to reconstruct across faults. The second example illustrates how a few individual horizon picks can improve the overall flattening result.

In Figure 2.22 is a 3D Gulf of Mexico data set provided by Chevron. The manually interpreted fault model is displayed in Figure 2.23. Two faults are identified in the figure. Fault 1 has part of its tip-line encased within the cube as can be observed by its termination in the time slice. Fault 2, on the other hand, does not terminate within the data cube. Because Fault 1 terminates within the data cube, no constraints need to be provided to flatten across it, however, Fault 2 requires some picking. In this case, I cross correlated one pair of traces across

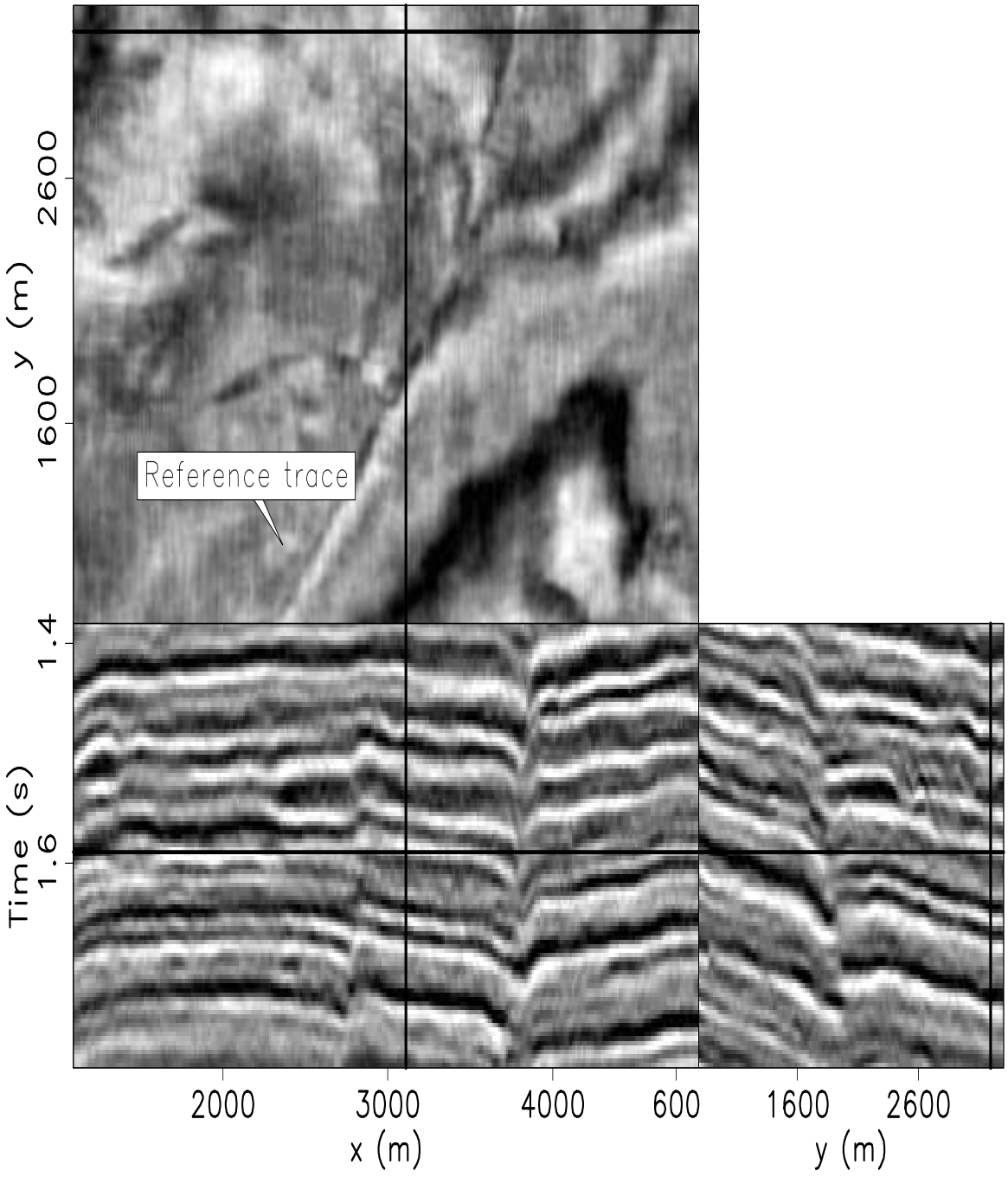

Figure 2.22: Gulf of Mexico data. The black lines superimposed onto the orthogonal sections identify the location of these sections: a time slice at time=1.584 s, an in-line section at y=3203 m, and a cross-line section at x=3116 m. The reference trace is located at x=2204 m and y=1259 m. flat-shoal_flt [CR]

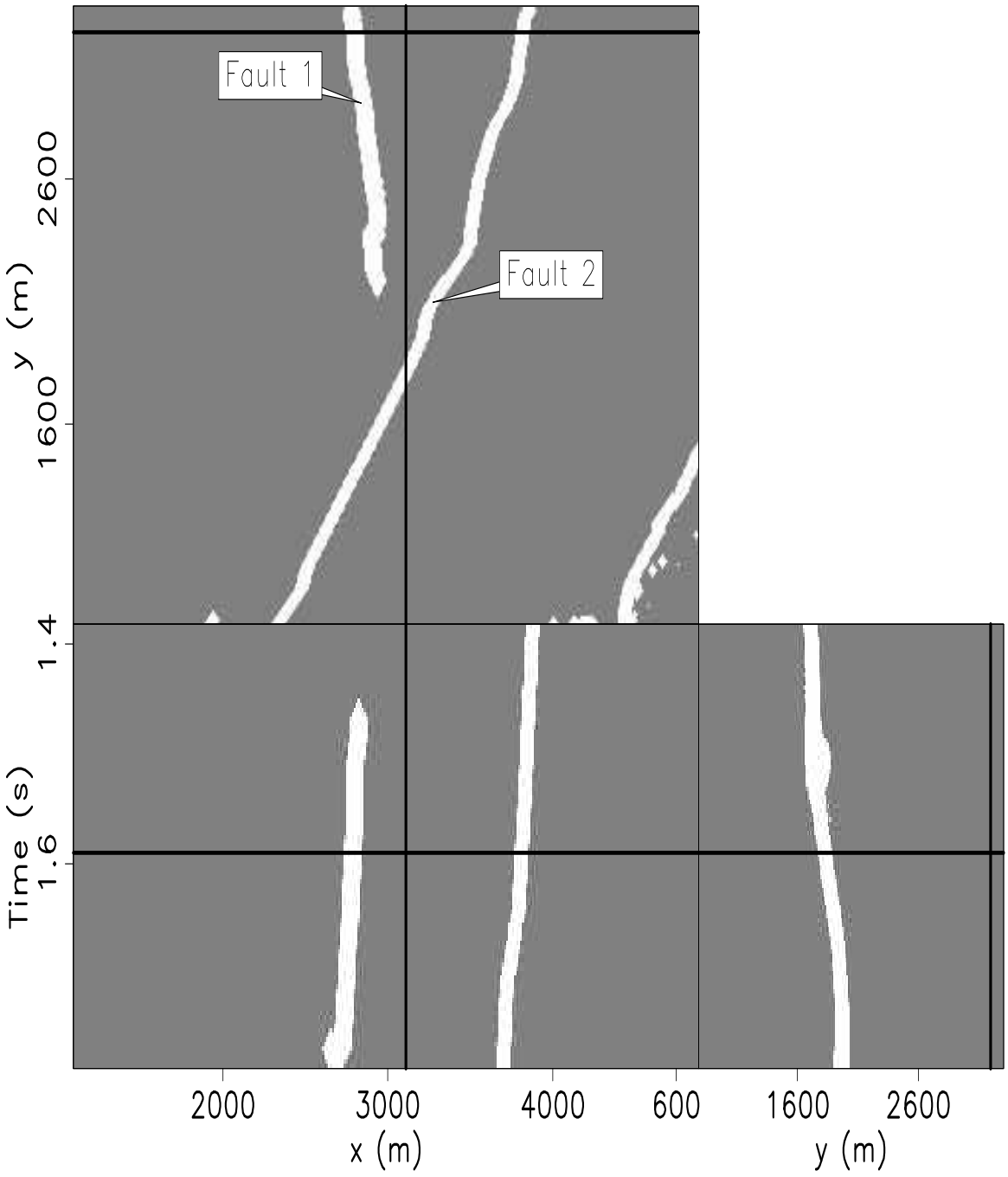

Figure 2.23: As Figure 2.22 displaying the manually picked fault model used for flattening. flat-shoal_flt_wt [CR]

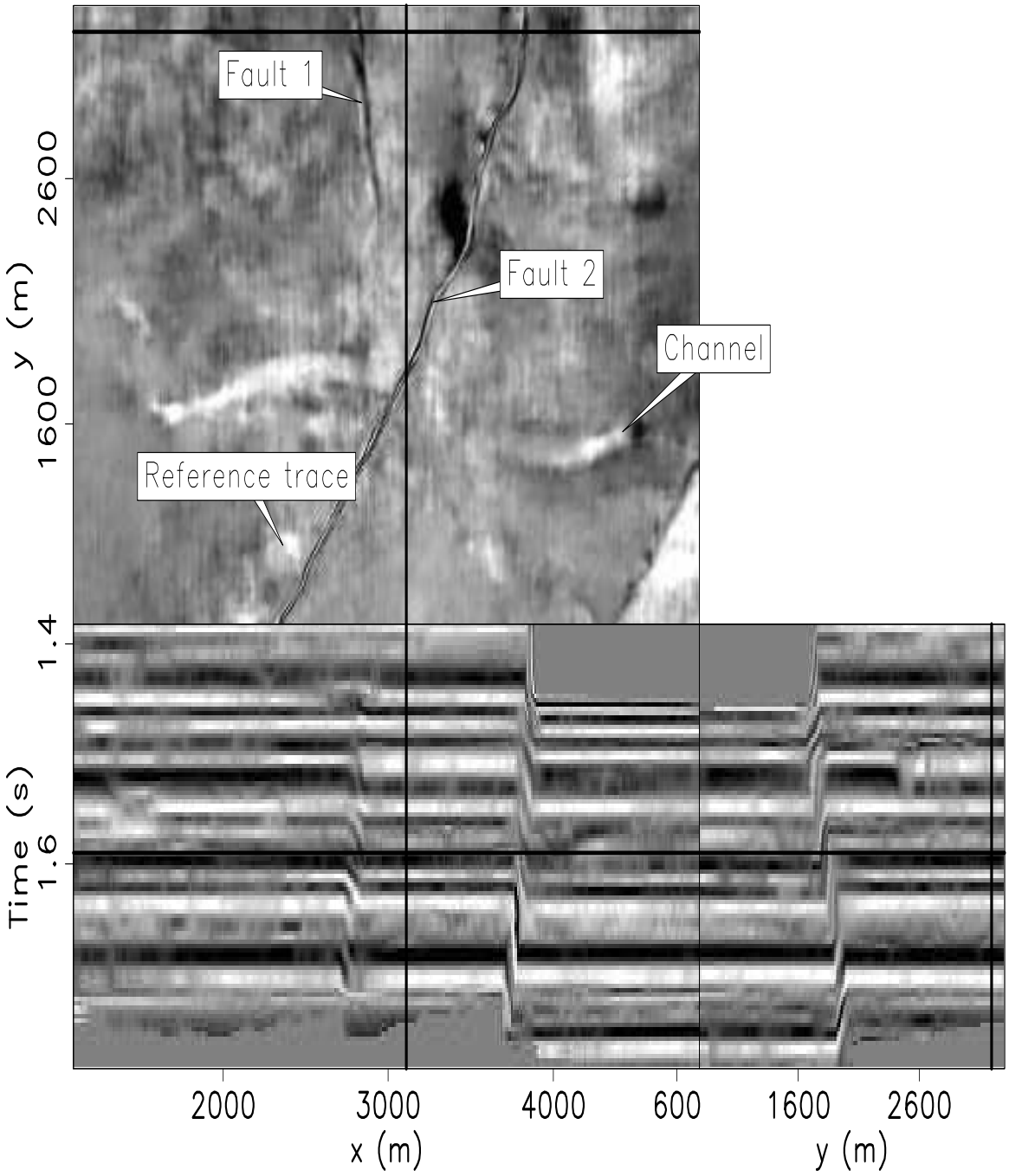

Figure 2.24: As Figure 2.22 only after flattening. Notice that reflectors on both sides of both faults are properly reconstructed. Also, notice a sinusoidal channel that is annotated on the horizon slice. flat-shoal_flt_flat [CR]

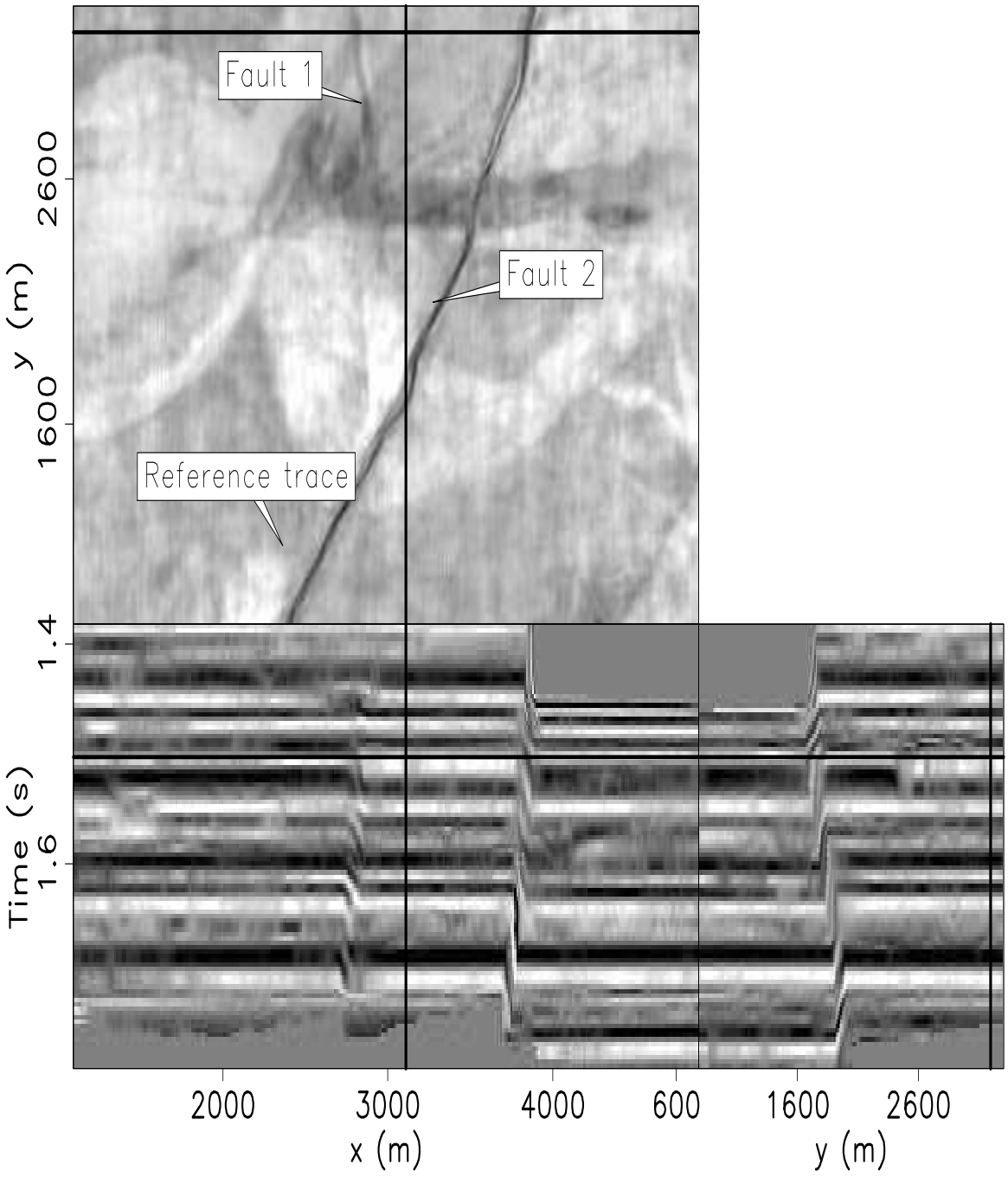

Figure 2.25: As Figure 2.24 except displaying a different horizon slice, time=1.504 s. Several stratigraphic channel features are visible. flat-shoal_flt_flat2 [CR]

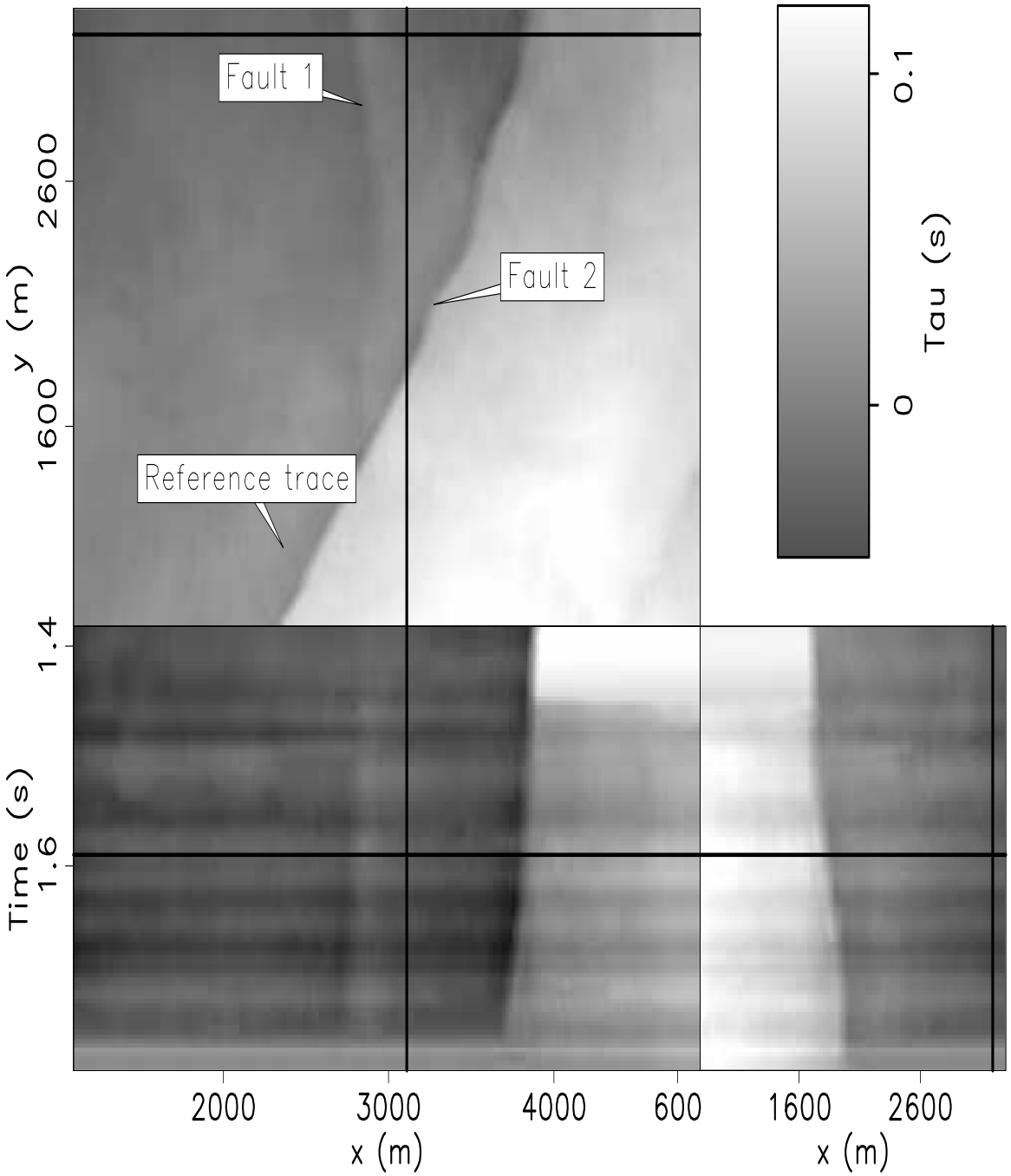

Figure 2.26: As Figure 2.22 showing the $\bar{\tau}$ field used for flattening. Recall that $\bar{\tau}$ is the time-shift field after the reference trace has been subtracted. The $\bar{\tau}$ field clearly indicates the locations of the faults. flat-shoal_flt_tau [CR]

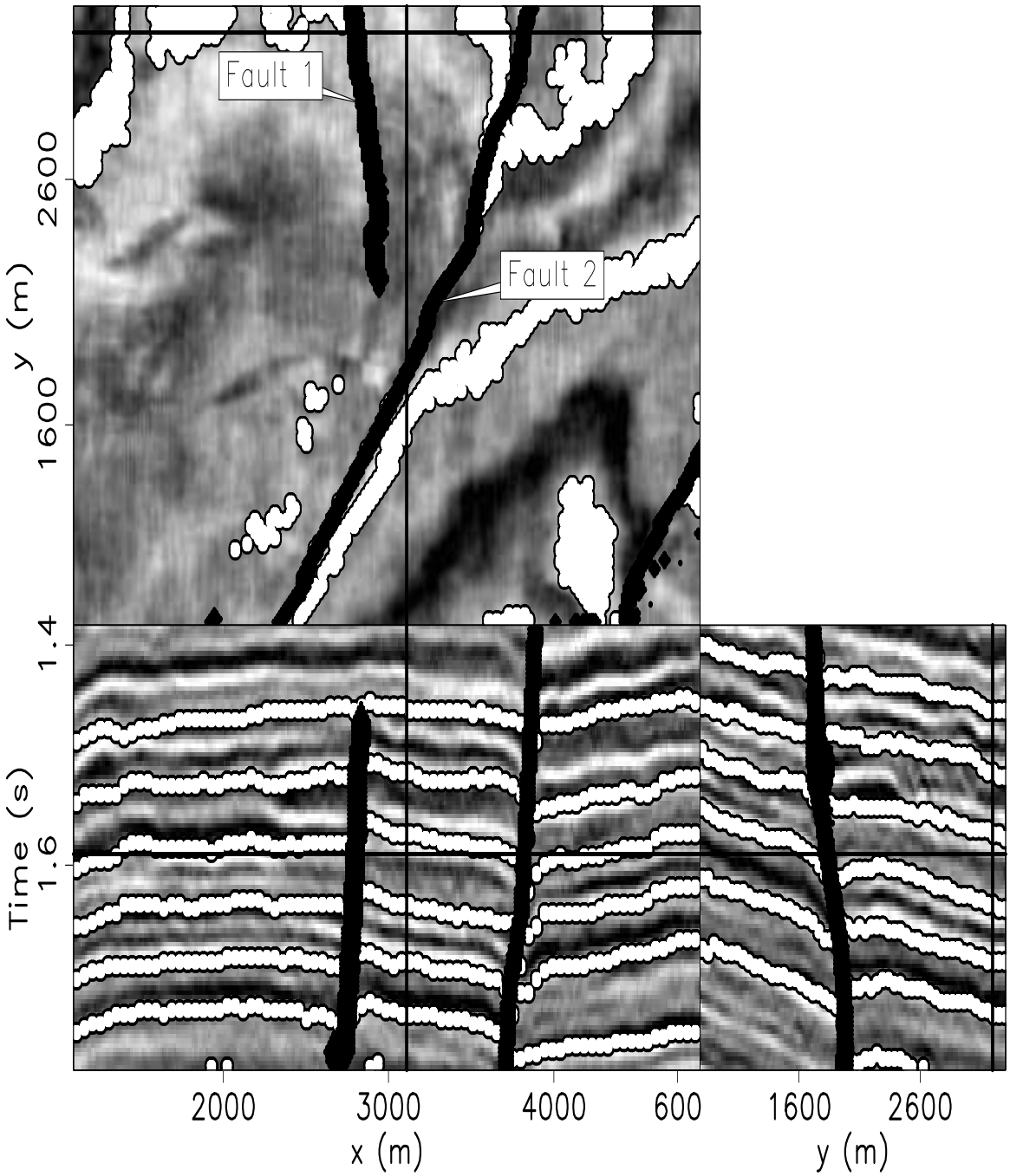

Figure 2.27: As Figure 2.22 only displaying every 25th tracked horizon of the Gauss-Newton constrained flattening method. The fault model (solid black) was manually picked. flat-shoal_flt_pck [CR]

Fault 2. Then I applied the weighted Gauss-Newton method with the weight $\mathbf{W}$ being the picked fault model. The binary mask $\mathbf{K}$ is ones for unconstrained model locations and zeros for constrained model locations. The initial model $\boldsymbol{\tau}_0$ (the picked pair of traces correlating across Fault 2) jumps at the fault by the amount of the lag of maximum cross correlation.

Figure 2.24 is flattened volume of the data in Figure 2.22. Notice the horizons are reconstructed across both faults. Notice the faint outline of a channel that is annotated on the figure. Another horizon slice of the same cube is displayed in Figure 2.25. Several stratigraphic features are reconstructed across both faults. In addition to a fault model, the only picks required were from a single trace of correlations across Fault 2. Also, the $\boldsymbol{\tau}$ field used to flatten this data is displayed in Figure 2.26.

In Figure 2.27 every 25th tracked horizon of the Gauss-Newton constrained flattening method in displayed. Notice the overlain horizons track their respective events across both faults.

In short, I reconstructed this 3D volume by correlating a single pair of traces across a fault. It should be pointed out that an automatic fault indicator could substitute for the fault model, reducing the amount of manual interpretation even further.

In Figure 2.28 is a 3D North Sea data set provided by Total. This data set has several issues including coherent noise and weak signal (see annotations). The flattening result is displayed in Figure 2.29. Several horizons that result from unconstrained and unregularized flattening are displayed in Figure 2.30. Without constraints or regularization, each horizon is flattened independently. Although I am only displaying five horizons, I have actually tracked all of the horizons in the cube. This is a key feature of flattening. Although the top two of the displayed horizons are well tracked, the lowest three are not. Figure 2.31 displays the τ field that results from the unconstrained, unregularized flattening. Notice that at approximately 1200 m depth there is a significant shift in the τ field. This marks where the salt boundary crosses the reference trace. The reference trace is located at the intersections of the panels at x=5040 m and y=3540 m. The salt interface is separating two regions of different dip and as a result is similar to an angular unconformity. The $\bar{\boldsymbol{\tau}}$ field is sensitive to the location of this boundary.

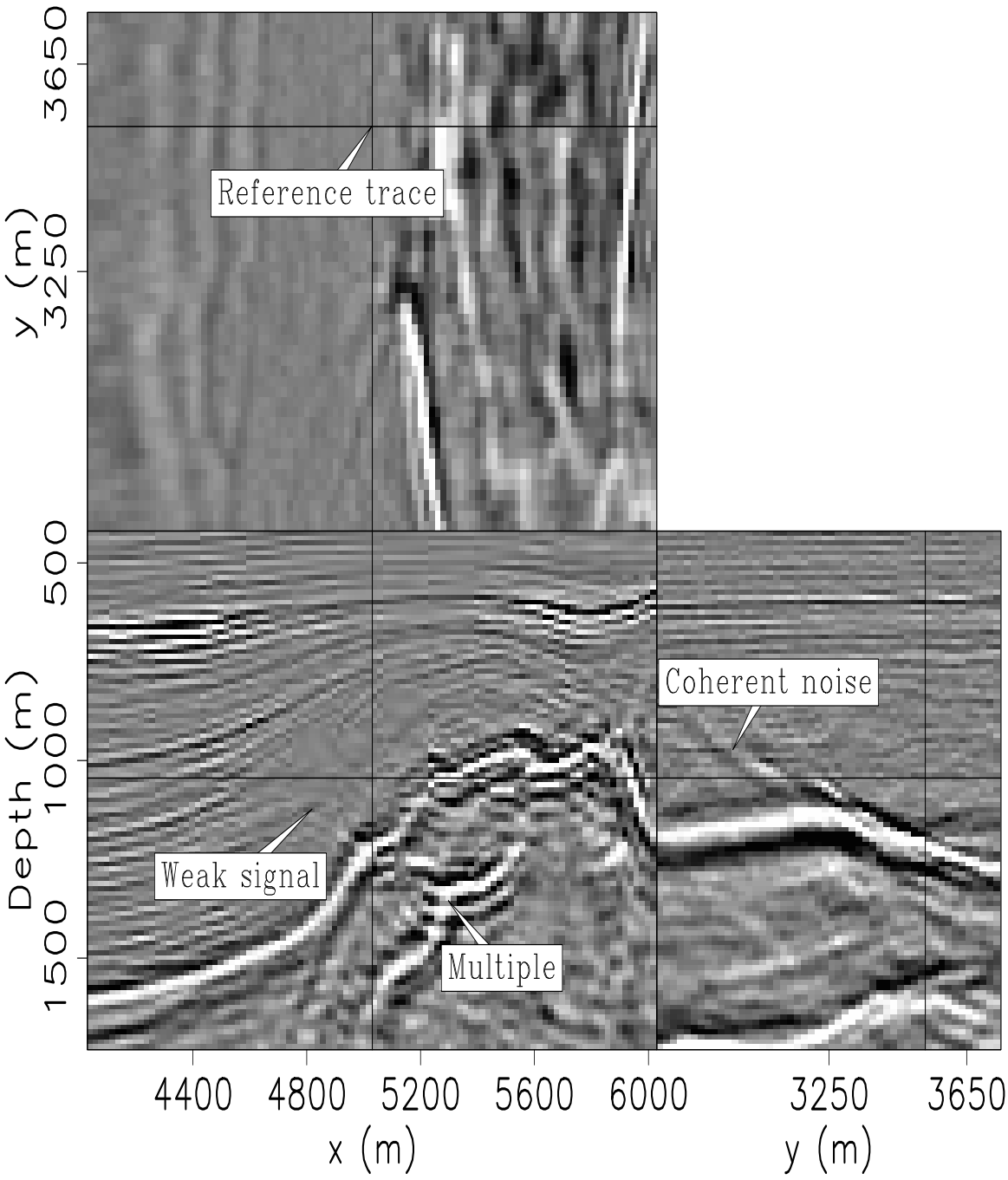

Figure 2.28: Total North Sea data. The black lines superimposed onto the orthogonal sections identify the location of these sections: a depth slice at time=1050 m, an in-line section at y=3540 m, and a cross-line section at x=5040 m. The reference trace is located at x=5040 m and y=3540 m, coincident with the intersection of the vertical sections. flat-elf_pck3 [ER]

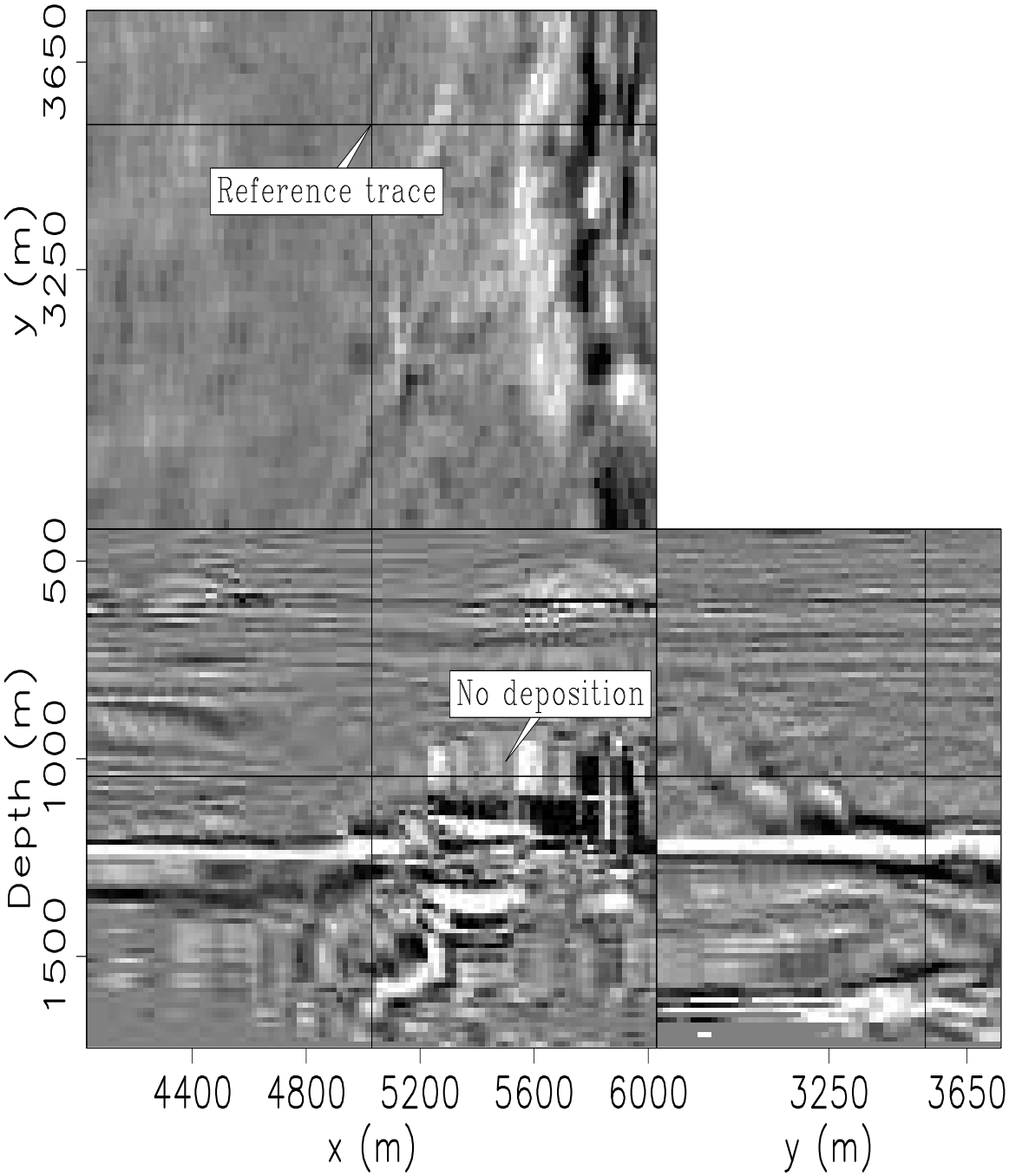

Figure 2.29: As Figure 2.28 only after applying unconstrained and unregularized flattening. flat-elf_pck3_unconstrained_flat [ER]

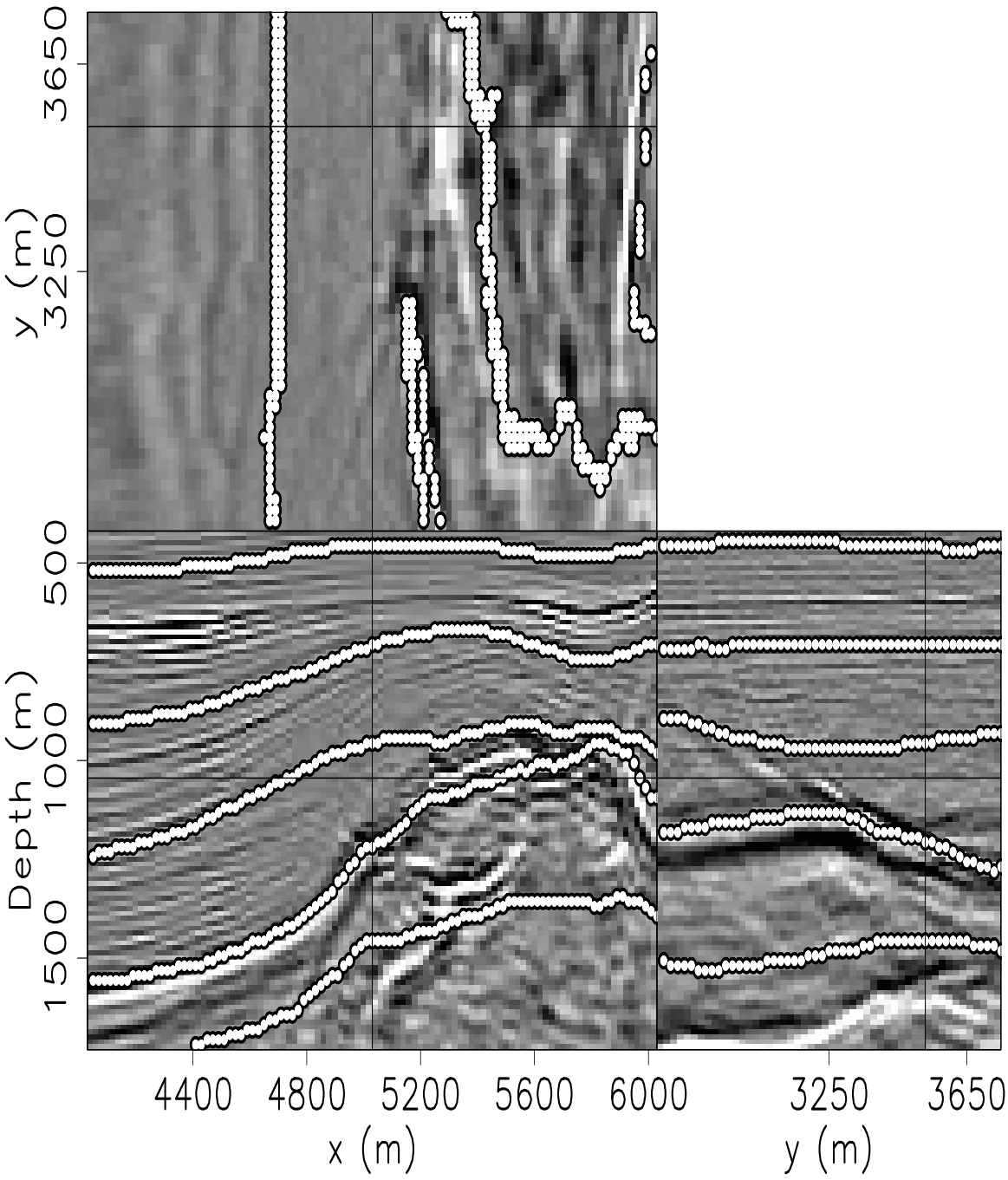

Figure 2.30: As Figure 2.28 except displaying five tracked horizons. Although I am only displaying five horizons, these flattening methods track all of the horizons in the data cube at once. This is an unconstrained solution. Notice that the third, fourth and fifth displayed horizons from the top are inaccurate. flat-elf_pck3_unconstrained_pck [ER]

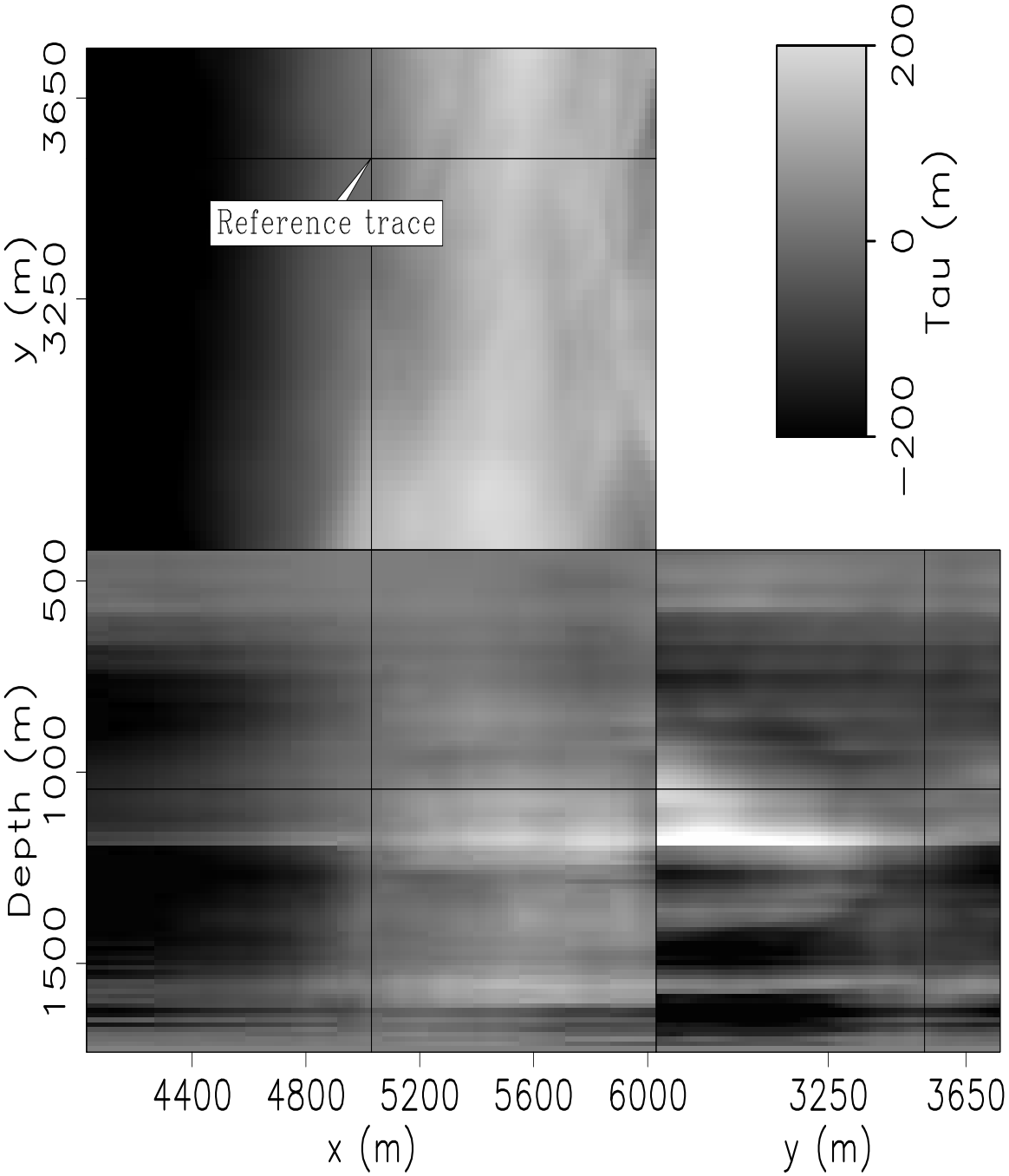

Figure 2.31: As Figure 2.28 except displaying the $\bar{\tau}$ field. Notice it is sensitive to the location of the salt boundary. flat-elf_pck3_unconstrained_tau [ER]

In Figure 2.32 is the result of flattening with regularization. To get this result I used a smoothing parameter $\epsilon = .7$ in equation (2.15). Because I enforced the conformity of adjacent horizons, the flattening result is smoother. However, an unwanted side affect is that the overall result is not very flat. This can be seen as well in Figure 2.33 where five horizons tracked using flattening with regularization are displayed. Although there is some improvement in the accuracy of the third horizon from the top, all of the other horizons are less accurately tracked than the unregularized result displayed in Figure 2.30.

To introduce constraints to this data, I selected the bright salt reflector that is very close to the fourth picked horizon in Figure 2.30. Since the fourth horizon is close to the bright salt reflector, it would have made an acceptable constraint but I decided to improve its tracking result by adding 22 manually interpreted points along the bright reflector. These 22 points are displayed in Figure 2.34 as black dots. To use these 22 points as constraints for the entire cube required two passes of the flattening algorithm. In the first pass, these individual points were passed to the flattening algorithm as constraints but without regularization ($\epsilon = 0.$). This was necessary because the sparsely picked points would not be honored if the full 3D volume was flattened with regularization because the effect of those sparse points on the residual of the entire cube is negligible. The resulting surface is also displayed in Figure 2.34. In the second pass the entire tracked horizon was passed as a constraint with regularization to get the result displayed in Figure 2.35. Notice that the bright salt reflector at approximately 1200 m depth is more flat than in the previous flattening results. Also notice that the horizons above (up to about 700 m depth) are better flattened. This is a result of adding the regularization, nearby reflections are forced to be more conformable to the constrained horizon. Unfortunately, several of the horizons, particularly near the top of the cube are not as flat as the unconstrained, unregularized result displayed in Figure 2.29. This is further supported by examination of the picked horizons displayed in Figure 2.36. Again, notice how the constrained horizon (fifth from top) tracks the salt boundary more accurately. Also, notice how the third displayed horizon shows higher quality tracking than previous results.

Thus far, different flattening results yield improvements in different areas of the cube. For instance, the unconstrained, unregularized results tracked the best in the shallower section where the signal-to-noise ratio is high whereas the horizons near the salt boundary were best

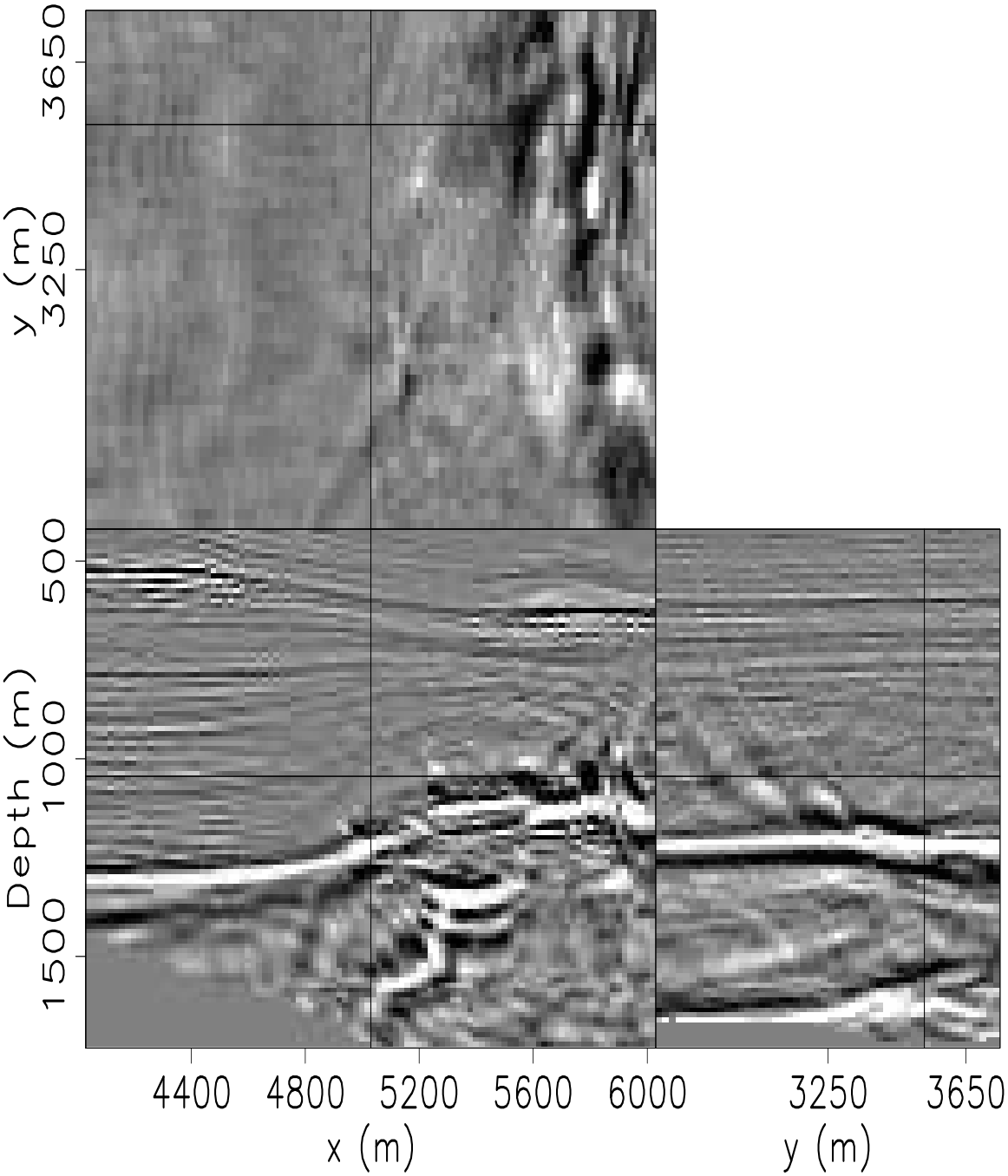

Figure 2.32: As Figure 2.28 only after flattening with regularization. Notice it is a smoother result than Figure 2.29 but overall less flat. flat-elf_pck3_reg_flat [ER]

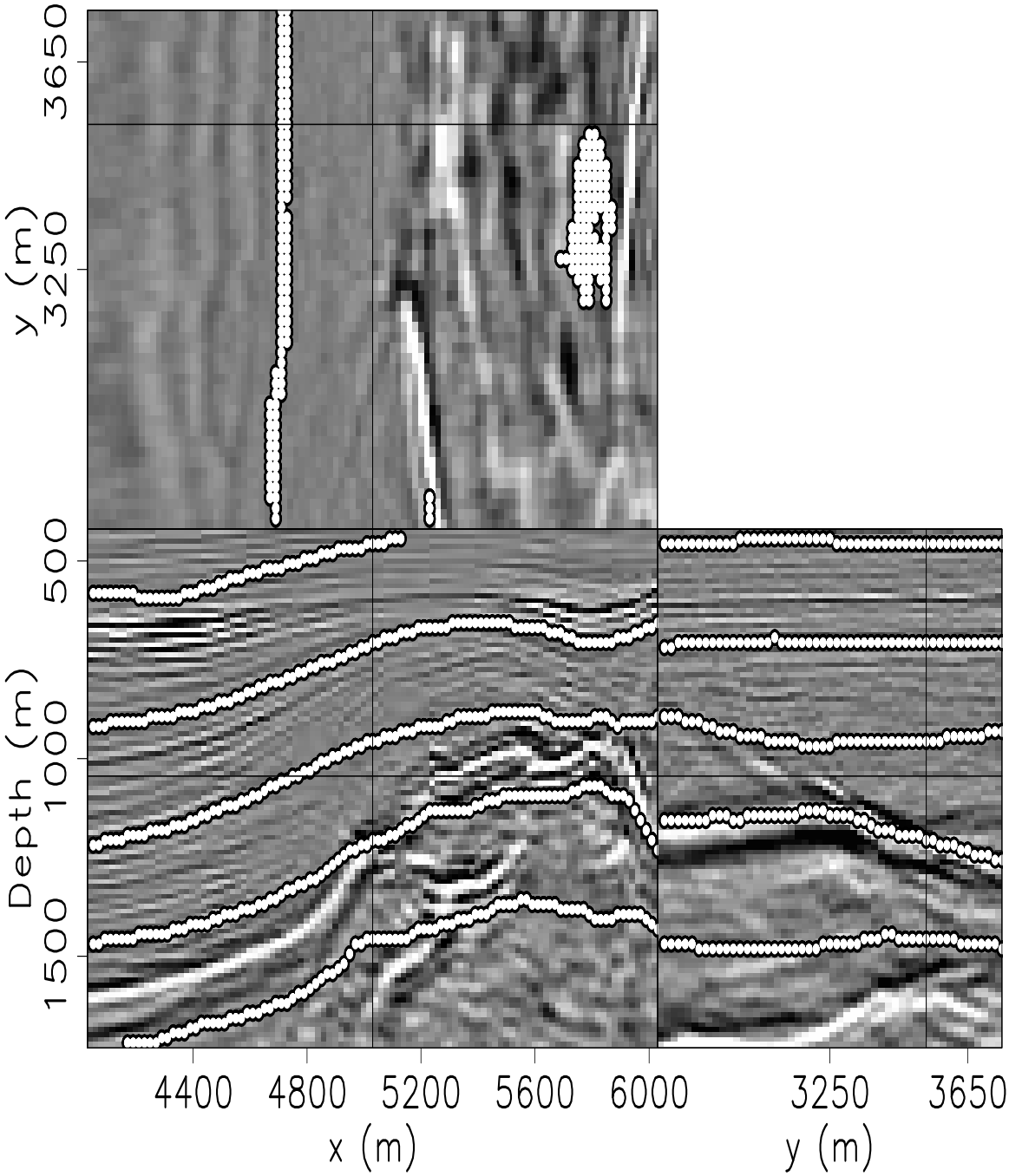

Figure 2.33: As Figure 2.28 except displaying five tracked horizons with regularization. Notice the third horizon is better tracked than in Figure 2.30 but the other four horizons are less accurately tracked. flat-elf_pck3_reg_pck [ER]

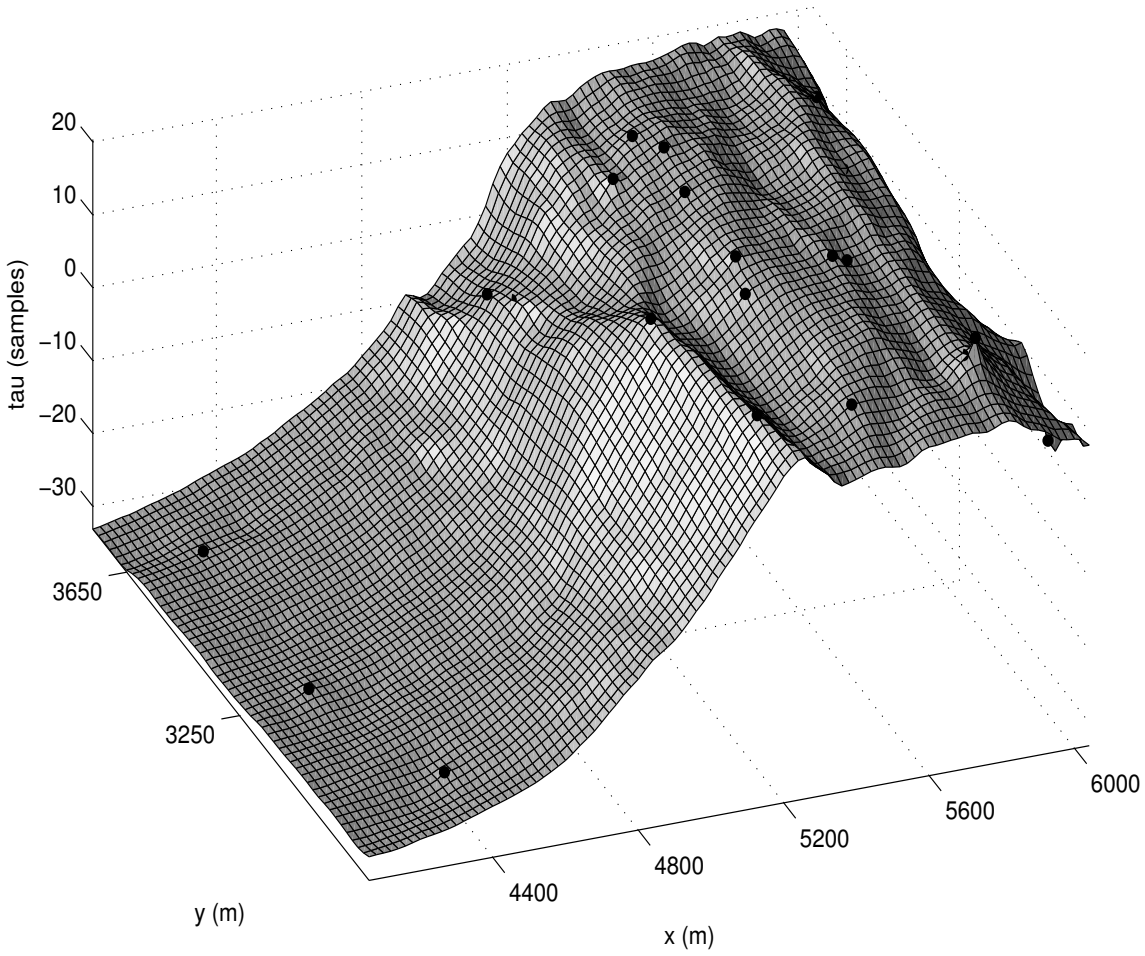

Figure 2.34: The 22 manually picked points and the auto-tracked surface using unregularized flattening with the manually picked points as constraints. This surface corresponds to the bright salt boundary in Figure 2.28. The value of this surface is zero where it crosses the reference trace. flat-tau_surf [CR]

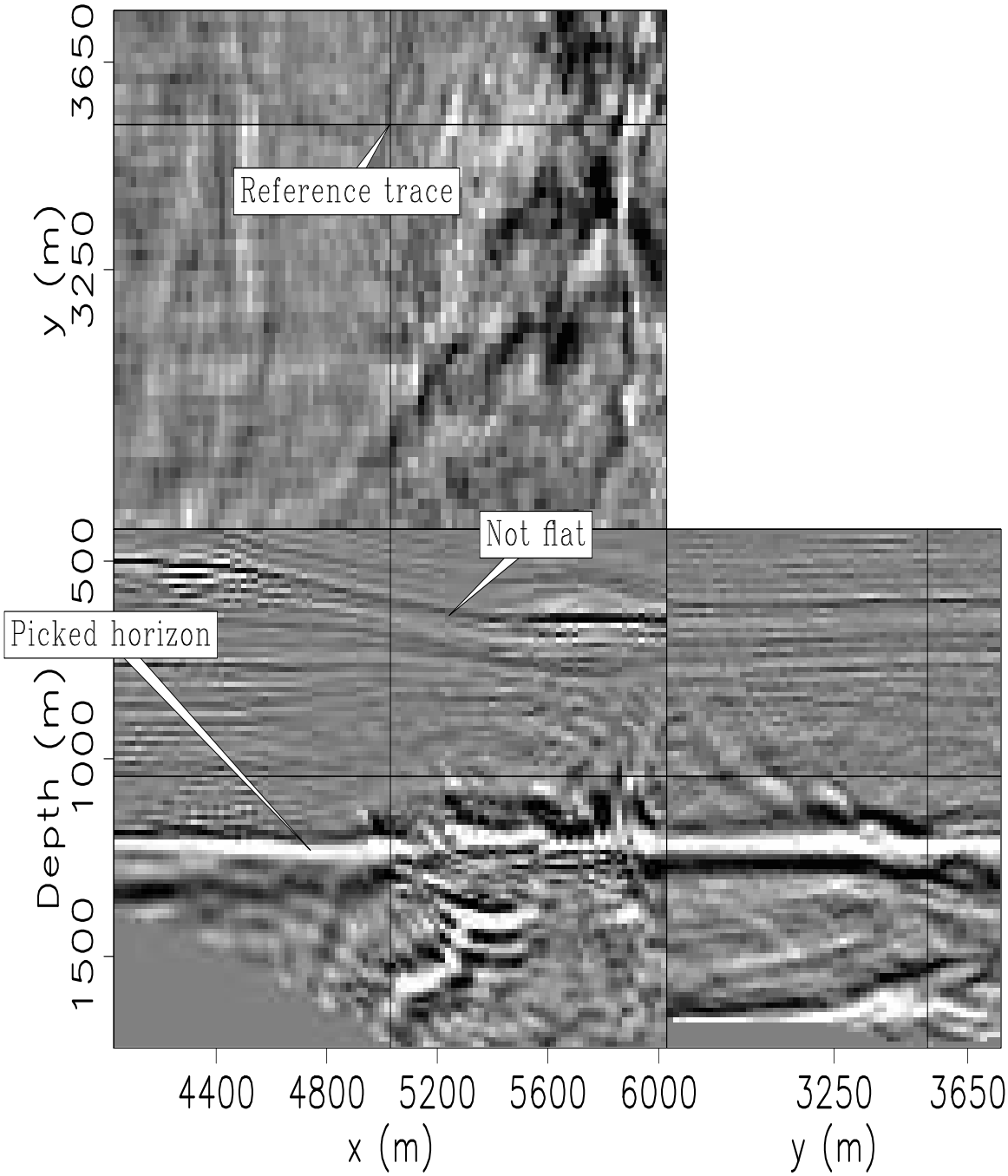

Figure 2.35: As Figure 2.28 only after flattening with regularization and the manually influenced salt boundary as a constraint. flat-elf_pck3_1con_flat [ER]

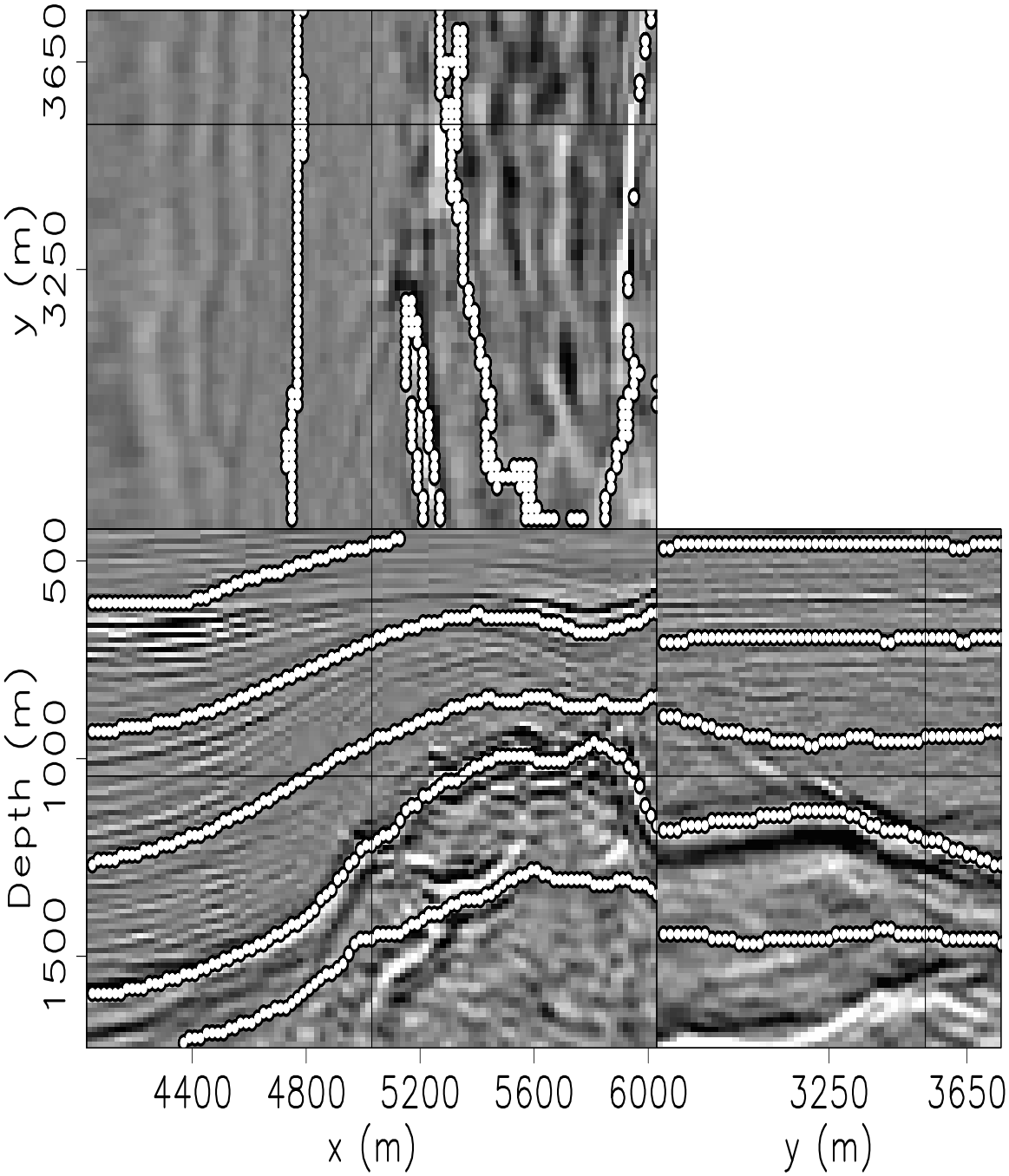

Figure 2.36: As Figure 2.28 except displaying five tracked horizons with regularization and the fourth displayed horizon as a constraint. The fourth horizon is the manually influenced salt boundary surface displayed in Figure 2.34. flat-elf_pck3_1con_pck [ER]

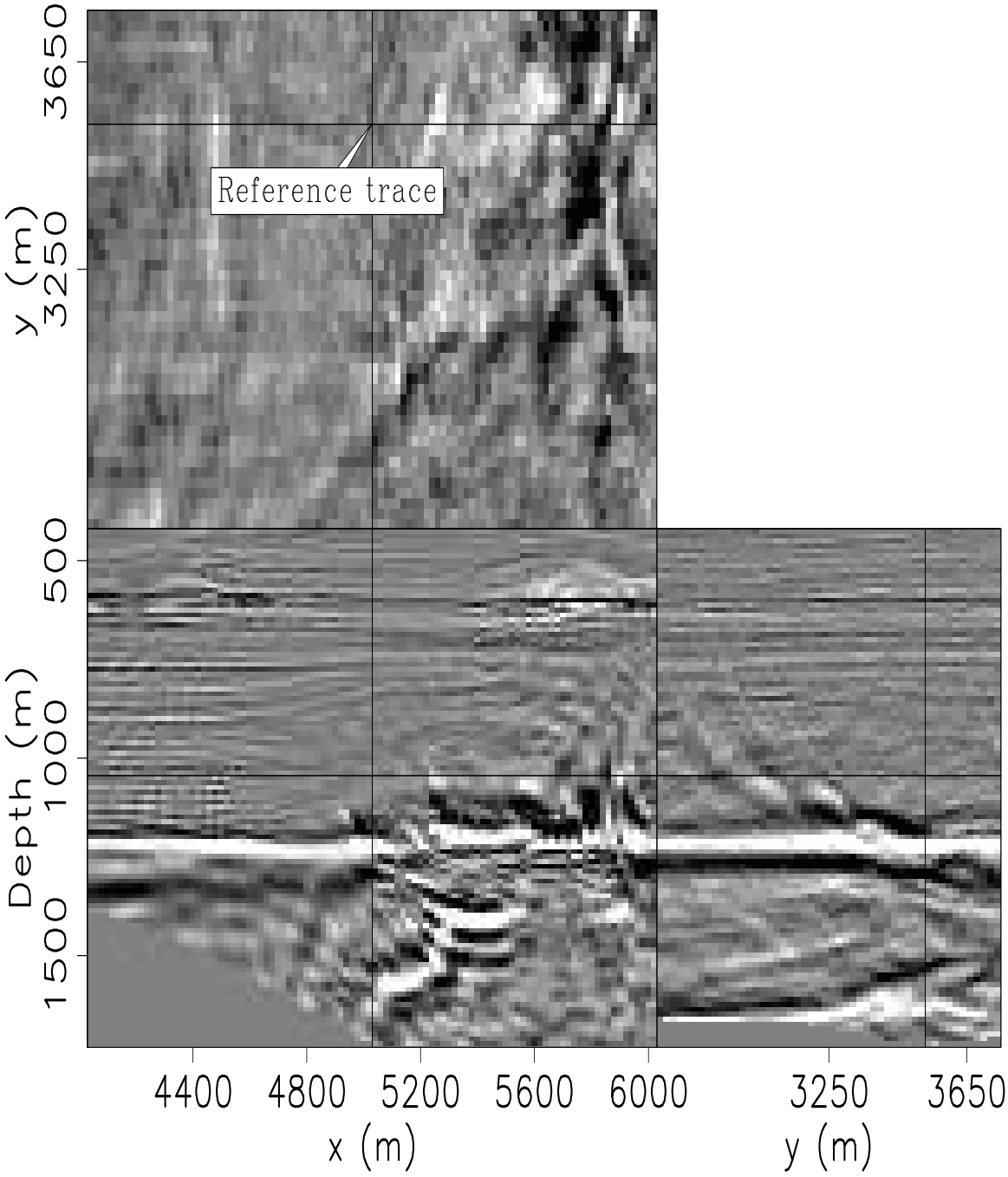

Figure 2.37: As Figure 2.28 only after flattening with regularization with constraints consisting of the manually influenced salt boundary and the upper portion of the unconstrained flattening result from Figure 2.29. flat-elf_pck3_many_con_flat [ER]

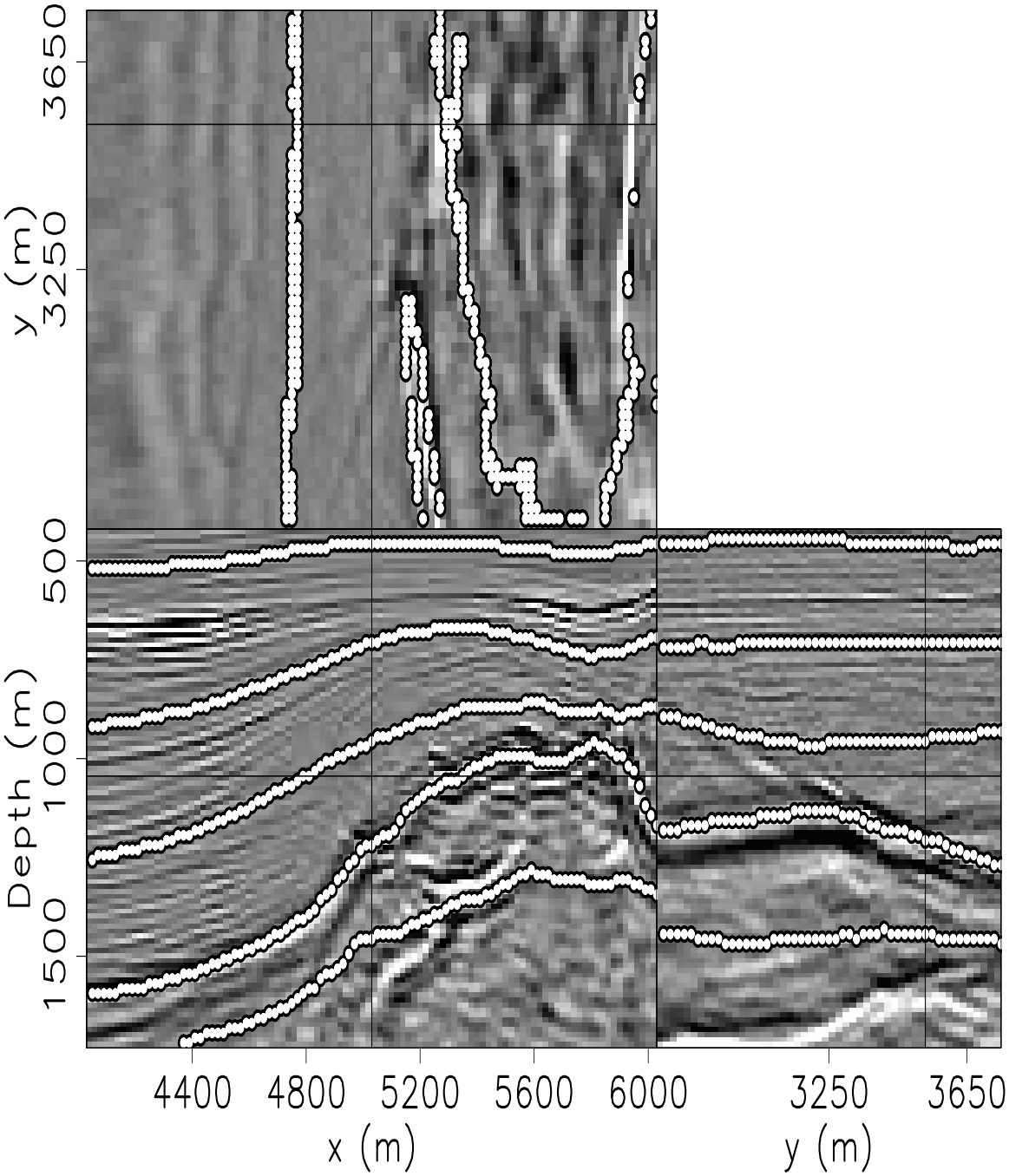

Figure 2.38: As Figure 2.28 except displaying five tracked horizons with regularization with constraints consisting of the manually influenced salt boundary and the upper portion of the unconstrained flattening result. flat-elf_pck3_many_con_pck [ER]

tracked in the constrained and regularized results in Figure 2.36. To combine the best of both worlds is straight forward. I merely pass the top part of the unconstrained, unregularized results as additional constraints. The result of this is displayed in Figure 2.37. Notice that now all of the horizons are well flattened. The picking result displayed in Figure 2.38 reveals that the first four reflectors displayed are accurately tracked. To improve the fifth reflector, I would likely need more manual constraints.

Iterative flattening

If I assume a Gaussian distribution of dips on each horizon, I can plot the mean and variance. Figure 2.39a shows the variance as a function of depth of flattened volumes with increasing amounts of geological constraints. Prior to calculating the variance, a short window AGC (10 sample window) was run to even the amplitude of the events. In a flattened volume, each depth step is associated with a separate horizon. The depths (or horizons) where the variance is low can be identified as good candidates for geological constraints.

From analysis of the variance of the amplitude as a function of depth, it is clear that the variance is relatively low in places where the unconstrained picking (Figure 2.30) was accurate and relatively high in areas where the picking was inaccurate. This could be used to automatically identify horizons that were accurately tracked on a first unconstrained pass. These horizons could subsequently be passed to the flattening algorithm as constraints to further improve the picking results of other reflectors. Unfortunately, there are also areas that are well tracked but still have high amplitude variance. For instance, the variance is high over the interval $580-600$ m in Figure 2.39a (uncon) but clearly well tracked in Figure 2.30. This can result from stratigraphic variation, hence areas of high variance are not necessarily poorly tracked.

It is also interesting to consider the statistics of dips re-estimated on flattened volumes. Plots of the variance of the dip in the x and y directions estimated on the previously described flattened cubes are displayed in Figures 2.39b and 2.39c. The variance of the re-estimated dip tends to be high where there was significant stretching and damage to the spectrum of the data by the flattening process. This can indicate areas of pinchouts or areas where the data

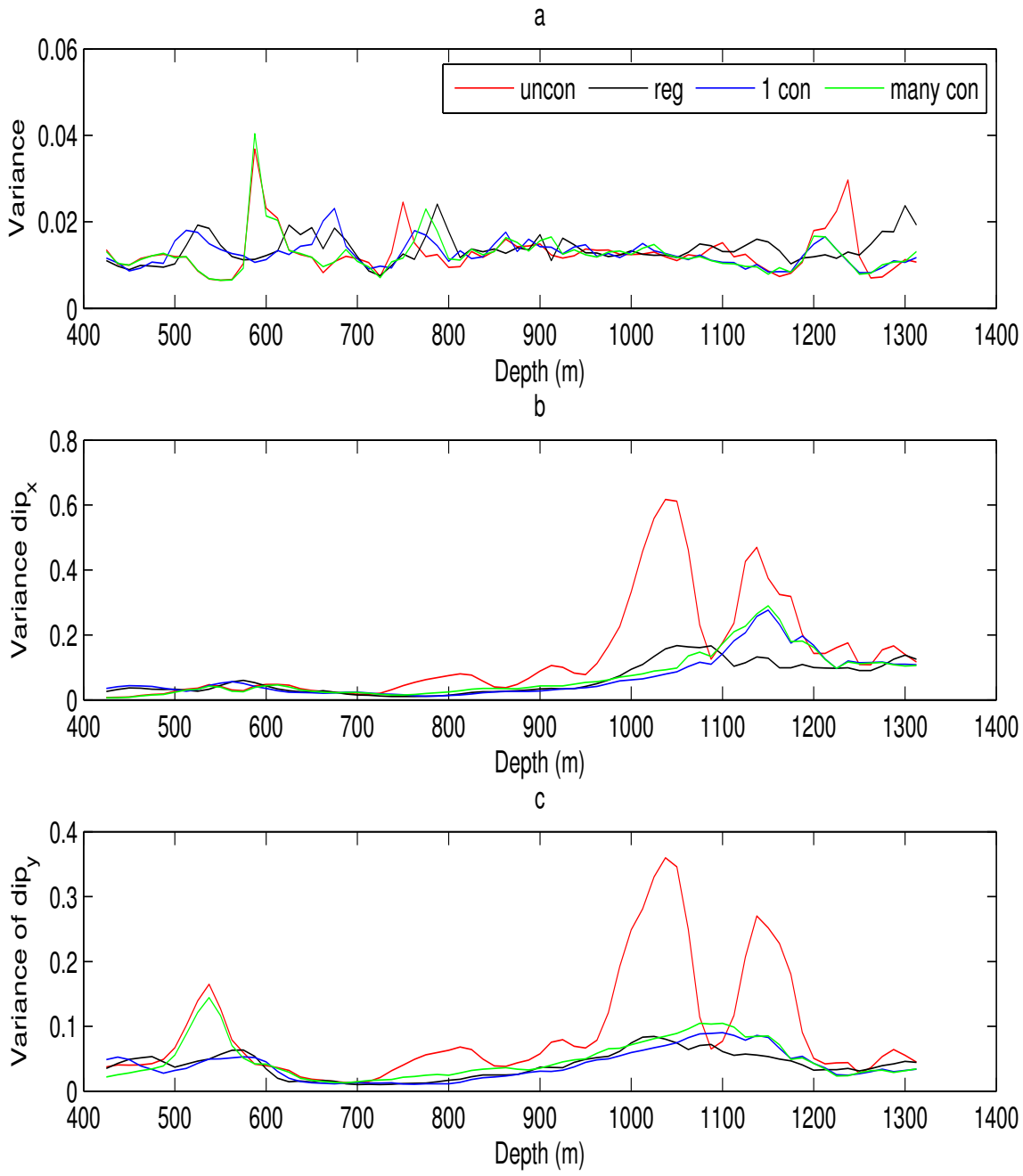

Figure 2.39: (a) The variance of the dip in the x direction re-estimated on each of the four different flattening results in Figures 2.29 (uncon), 2.32 (reg), 2.35, and 2.37 (many con). (b) The variance of the dip in the x direction re-estimated on each of the four different flattening results. (c) The variance of the dip in the x direction re-estimated on each of the four different flattening results. flat-stat_combo [CR]

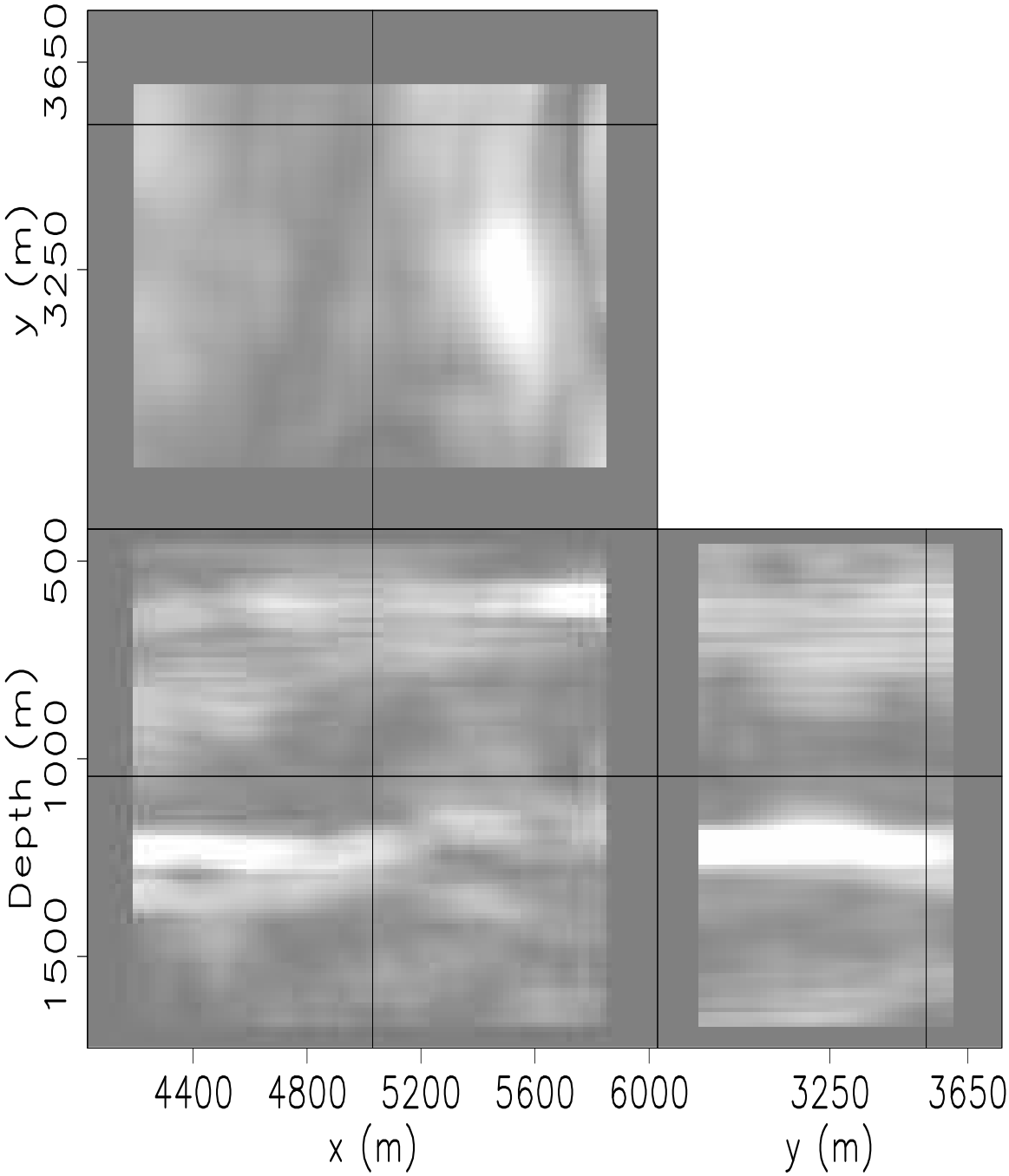

Figure 2.40: Local energy calculated on the unconstrained flattening result in Figure 2.29. The gap on each side is the half-width of the stacking window. flat-elf_pck3_uncon_energy [CR]

quality has resulted in severe mis-picking. The high variance of the re-estimated dips for the unconstrained flattening result seems to correspond to the stretching artifacts between 950 and 1050 m in Figure 2.29.

Local stack energy maybe useful to determine what portions of cubes are sufficiently flat. AGC (window=10 samples) was first applied to the unconstrained flattening result in Figure 2.29. The energy of stacked local windows (6 X 16 X 12 samples) is displayed Figure 2.40. The dim areas roughly correspond to locations that are not well flattened. Also, the shallow bright areas correspond to locations that are well flattened. This could possibly be used to automatically determine which parts of flattening results could be used as constraints into subsequent iterations.

An overall measure of the quality of the flattening is the total stacked energy. Once a cube is flattened, it can be summed into a super trace. The total energy of this super trace will likely be greatest for the flattest cube. The energy for the four flattened cubes with different amounts constraints is displayed in Figure 2.41. Notice that unconstrained, unregularized cube (uncon) has greater energy than the regularized cube (reg). This makes sense as the regularized cube doesn't appear to be too flat. Next, adding a single constraint (1 con) increases the stack energy. Finally, the stack of the flattened cube with many constraints (many con) in Figure 2.37 has the greatest energy. Accordingly, by inspection alone, this cubes seems to be the overall flattest.

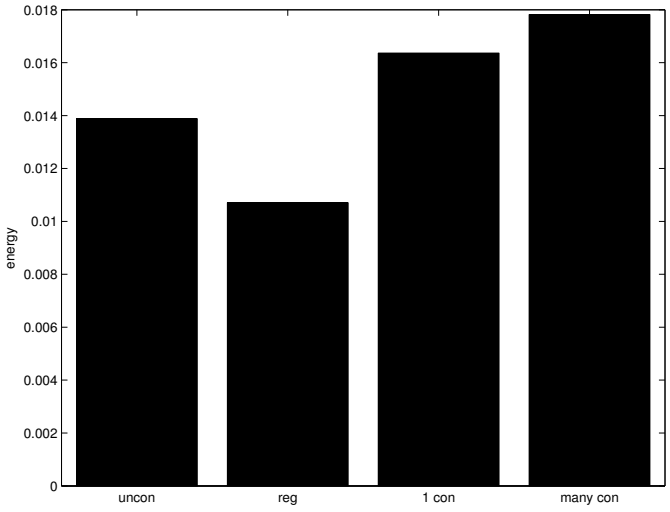

Figure 2.41: Energy of stacked traces of the four different flattening cubes in Figures 2.29 (uncon), 2.32 (reg), 2.35, and 2.37 (many con). Notice the greatest stack energy result is from the overall flattest cube (many con). flat-total_energy [CR]

FLATTENING APPLICATIONS

The flattening process has numerous post-stack applications. The ability to convert a data cube from time (or depth) to a horizon cube has obvious and immediate interpretive uses by simply presenting the data in a more geologically meaningful way. Additionally, the mapping from unflat to flat ($\boldsymbol{\tau}$ field) contains all of the information of the shapes of all of the reflectors in the data cube. This has potential ranging from isopach analysis (Lomask et al., 2005) to horizon shape attribute volumes. In the case of flattening with faults, the flattening process automatically captures the slip distributions (fault contours) along the faults. This leads naturally to fault contour analysis which has numerous applications originally described in Lomask (2002). Lastly, knowledge of the combination of fault slip and horizon deformation can be related to local stress measurements.

The prestack applications of flattening are also numerous. The most obvious being gather alignment. Additionally, the $\boldsymbol{\tau}$ field can be used for tomographic velocity analysis (Guitton et al., 2004). Furthermore, the $\boldsymbol{\tau}$ field can be used to fit parametric curves from angle gathers for residual moveout velocity updating. Flattening can also be used to detect and quantify migration artifacts. Although still largely under-exploited in prestack applications, flattening has the ability to handle any number of dimensions.

In this section, I will describe several applications of flattening. At this writing, many of these applications have not yet been tested. Because the overall thrust of the thesis is interpretive related, these applications are primarily post-stack in nature.

Fault contour applications

A flattened cube with faults that is properly reconstructed implies that the slip distribution along the fault surfaces have been captured within the $\boldsymbol{\tau}$ field. Factors governing the slip distribution include the stress field, proximity to other faults, rock strength, tectonic history, and loading rates (Pollard, 2001). If the slip distribution were easily and quickly obtained, it could be added to interpretation and processing work-flows.

In Figure 2.26 is the $\bar{\boldsymbol{\tau}}$ field used to flatten the data in Figure 2.22. Notice that it changes

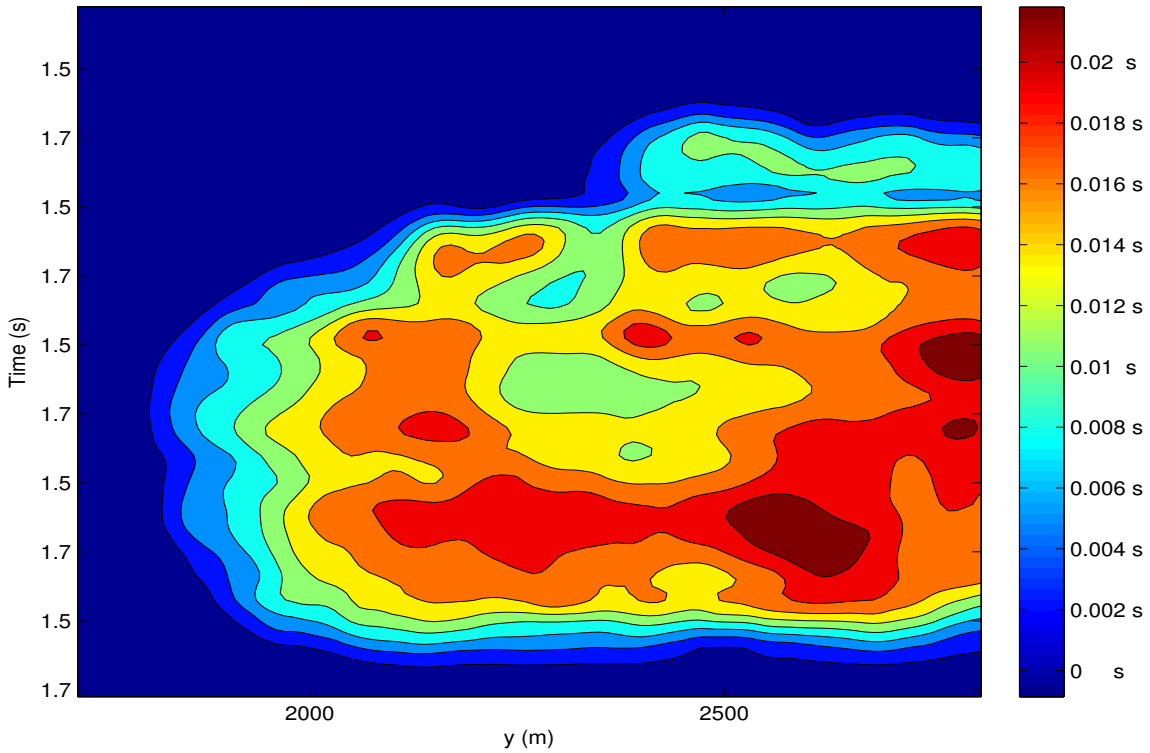

Figure 2.42: Fault contours from Fault 1 in Figure 2.22. The slip is measured in time. These were created by extracting and differencing the $\boldsymbol{\tau}$ field in Figure 2.26 from both sides of the fault. flat-fault_cont_m [CR]

in magnitude abruptly across the major fault, referred to as Fault 2. This is because the beds are shifted significantly from each other across the fault. The minor fault, Fault 1, can also be seen in this cube as well. Recall that this cube was flattened with a fault model and only one correlated pair of traces across Fault 2. Nothing more than a fault model was required to flatten across Fault 1. I extracted the $\bar{\tau}$ field values from either side of Fault 1 and differenced them. The contoured difference is shown in Figure 2.42. The contours show the vertical slip in time across the minor fault, Fault 1.

Below, I outline several applications of fault contours, originally presented in Lomask (2002).

Stresses and Rock Strengths First of all, one can speculate about the interplay of rock strengths and stresses. The contours are a vital input for researchers studying fault mechanics. They help unravel the stress and strain fields. Understanding the slip distribution can imply slip and position of neighboring faults.

Seals One can use fault contours to see where the minimum displacement should be to insure an adequate seal. For instance, if the thickness of a sand reservoir is known, then you will likely have a shale on sand seal if the displacement is greater than the thickness. In addition, in building a reservoir model, faults which have insignificant displacement would not necessarily be included.

Geological interpretation Combining the contours with other data can help unravel the geological history by revealing periods of growth along the faults in the case of syndepositional faulting. Periods of rapid fault growth can be a result of increased sediment loading that can be tied to mountain building events, eustatic fluctuations, and climate changes.

Well-ties in 3D modeling The slip from a fault contour can be back interpolated at well locations. In the case of normal faulting, the magnitude of the slip can be compared with the missing section from the wells. In the case of reverse faulting, it can be compared with the repeat section. This information can be incorporated as an additional well tie and input into 3D models.

Constrain processing By tying with the regional stress field, likely faulting strike, dip and magnitude can be estimated. This has potential to help constrain velocity analysis. In addition, fault displacement information from well data can be used as constraints.

Missing section information used in standard fault interpretation Knowing the fault displacement can be incorporated into an interpretation program to help correlate faults.

Local stress volumes

Maerten et al. (2002); Maerten and Maerten (2002) describe how flat simple models created with some basic knowledge about the regional stress field and rock properties can be deformed or forward modeled until agreement with the structure gleaned from seismic data is found. This involves deforming the layers and faults together. Once a match is achieved then they are able to create maps of the local stress field which can then be related to secondary fractures and faults caused by folding. Therefore, it seems like a straight forward application of flattening to incorporate the $\boldsymbol{\tau}$ field into an inversion process that creates local stress volumes.

Unconformities

It seems obvious from inspection of the $\bar{\boldsymbol{\tau}}$ field in Figure 2.31 that the $\bar{\boldsymbol{\tau}}$ field contains information of the location of the salt boundary. This is because the salt boundary is similar to an angular unconformity, separating two regions of different geological dip. Similarly, inspection of the $\bar{\boldsymbol{\tau}}$ fields in Figures 2.13 and 2.16 reveals the same effect yet this time with a sedimentological angular unconformity. This $\bar{\boldsymbol{\tau}}$ field was created by flattening the data in Figure 2.11. Besides using the stretching of the data, another way to highlight the unconformity(or pinchouts in general), I can unflatten a volume filled with 1s. Where multiple 1s are mapped to the same location, I will add them. The resulting volume is displayed in Figure 2.43. This is a cube highlighting areas where horizons are overlaying one another. In other word, it is highlighting pinchouts. The brightest pinchout is associated with the major angular unconformity near 2000 m depth. It should be pointed out that the results are sensitive to the location of the reference trace. If the reference trace is placed in the thickest section then it will highlight the

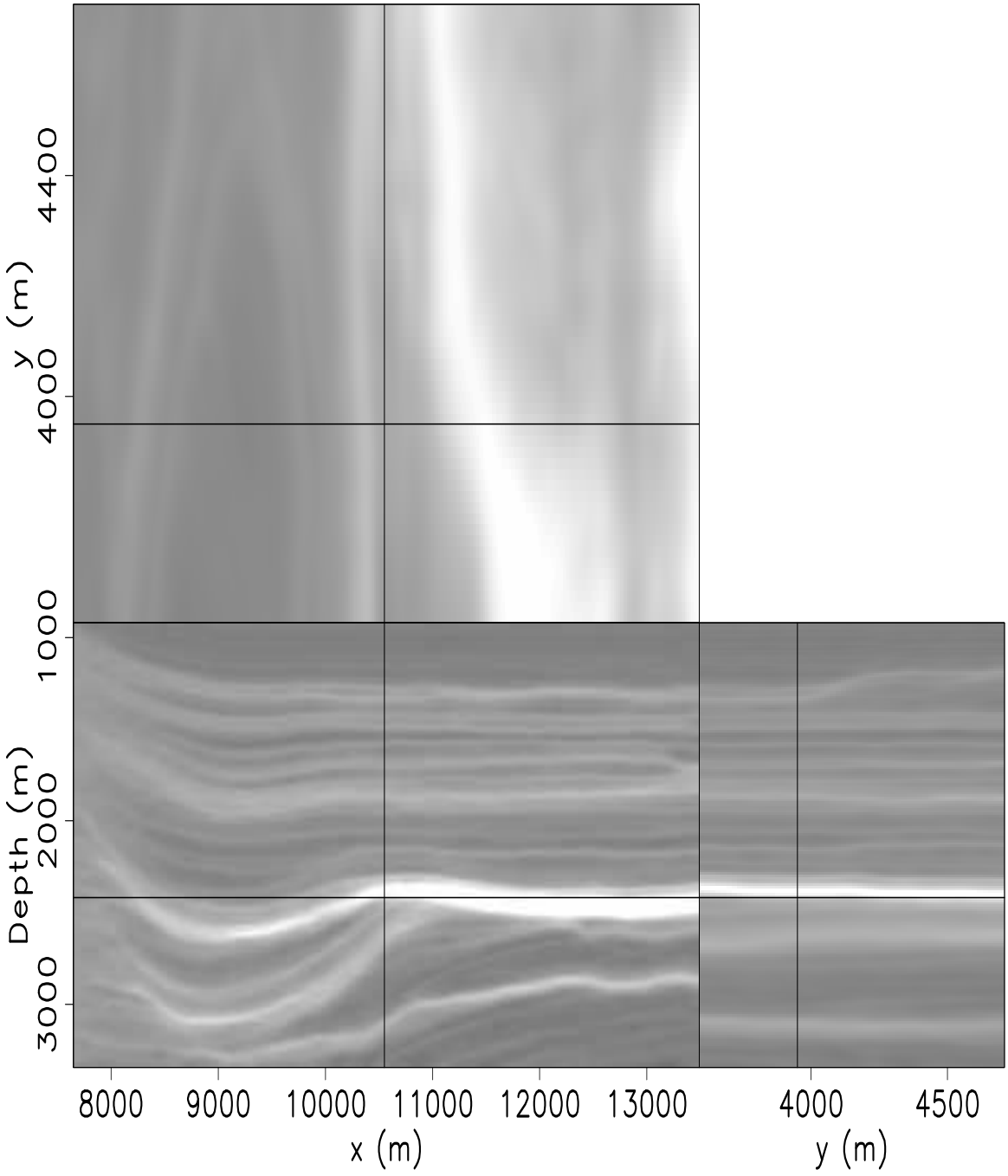

Figure 2.43: The result of unflattening a cube of all 1s. This highlights areas where horizons come together (pinch-out) and is an indicator of unconformities. flat-elf2_unconform [ER]

pinchouts. If the reference trace is in the thinnest location, then it will not successfully identify the pinchouts. In the next section, another approach to address this limitation, by finding a different reference location for each horizon, is described.

CONCLUSIONS AND DISCUSSION

I present a method to efficiently and robustly flatten 3D seismic cubes. This method uses an efficient implementation via the discrete cosine transform within a non-linear iterative scheme. I demonstrated its effectiveness on both synthetic and a series of field data sets with varying degrees of structural complexity.

Data cubes with vertical, laterally limited faults can be flattened by applying a residual weight. This allows horizons to be tracked around the faults. The weight can be created from a previously determined fault model or coherence attribute. The weight merely identifies inaccurate dip values estimated at fault discontinuities so that they will be ignored within the inversion. If not provided by a previous fault model, the residual weight can be estimated automatically by extending the basic flattening method to use iteratively re-weighted least squares (IRLS). In this case, the flattening method can be thought of as a fault detector that generates the fault picks that best flatten the data. Unfortunately, this IRLS scheme adds significant noise sensitivity and computation time to the basic flattening method described in this paper.

Although all of the flattening results presented in this paper were generated using a single reference trace, for many geological applications it makes sense to identify an optimum reference location for each horizon. In order to preserve all of the data, the optimum location should be chosen to go through the thickest geological section. The flattening algorithm can be adapted to find this location by applying 2D ((x, y)-plane) flattening for the two horizons associated with the top two time slices. The new reference location is the thickest location that is determined from differencing the resulting $\boldsymbol{\tau}$ fields. This processes is repeated using the new reference location until the thickest location coincides with the reference location. Then the next pair of horizons is flattened using the new reference location. This process is repeated throughout the data cube.

In some cases it may be better to flatten to a mean-dip plane instead of flattening to a horizontal plane. This flattening method operates by shifting the data only vertically. By doing this, the lengths of horizons are not preserved. The magnitude of this is a function of the cummulative structural dip. By flattening to a mean-dip plane, the horizons are distorted less. This is akin to first rotating the original data cube to that it is as flat as possible and then apply the flattening technique.

As demonstrated here, the ability to incorporate some picking allows the reconstruction of horizons across faults that cut across the entire data cube. An interpreter can pick a few points on a 2D line and then flatten the entire 3D cube. With computational improvements in both the algorithm and hardware, this method could be applied on the fly, as the interpreter adds new picks.

Also, as pointed out in Lomask (2003a), geological features can be interpreted on the flattened horizon sections then subsequently unflattened into their original structural shape to tie with wells and other data.

The ability to use flattening in an iterative scheme is still under-exploited. Once a data cube has been flattened, dips can be re-estimated on the flattened cube and then flattened again using the statistics of the dip as a measure of flatness. It seems plausible that this approach has the potential to greatly improve the quality of the flattening process. Furthermore, better flattening results can be obtained by first flattening low-passed versions of the data and gradually adding in higher frequencies.

The obtained estimate of $\boldsymbol{\tau}(x,y,t)$ has many potential uses; e.g., the proposed method can easily be adapted to flatten data cubes on one or any particular combination of horizons. This would assist geologists in analyzing thicknesses for rates of deposition and timing of structural events in growth faults. Furthermore, for faulted data sets that have been successfully reconstructed by this flattening method, the integrated time shifts contain the slip distributions along the fault planes.

Ultimately, the quality of the flattening result depends on the quality of the input dips. Dip estimates can be affected by noise. Using weights can remove the effect of some inaccurate dips, however there must still be enough good quality dips to capture the general trends of the

structure. Furthermore, crossing events cannot be easily flattened by this method.

Chapter 3

Image segmentation

INTRODUCTION

Salt interface picking is a difficult and time consuming component of constructing a velocity model. Although the amplitude of reflections from salt boundaries is often bright, traditional amplitude-based autotrackers regularly fall prey to local discontinuities and amplitude variations, both of which are common traits of salt boundaries. Consequently, many hours of manual editing are often required. This is compounded by the iterative process of updating the velocity model and re-migrating the data. Other tracking techniques such as artificial neural networks are less sensitive to amplitude variations but are still prone to error if the seismic wavelet character varies significantly from the training data (Leggett et al., 1996). Brown et al. (2006) tackle picking from a semi-global optimization perspective but between seed picks which is somewhat cumbersome in 3D. A modified version of normalized cuts image segmentation (NCIS) (Shi and Malik, 2000) provides an alternative picking method that looks for the best overall 3D picking result via a global optimization problem and is thus less sensitive than traditional autotrackers to local discontinuities.

NCIS (Shi and Malik, 2000) is a pattern analysis technique developed to extract global impressions from images. Global impressions are coherent groups of pixels that may, for example, represent the pixels of an image in the foreground while excluding pixels in the

background. Hale and Emanuel (2003, 2002) introduced this technique to the seismic industry by painting 3D atomic meshes. Lomask and Biondi (2003) and Lomask et al. (2004, 2006a, submitted 2006) subsequently applied the method to the problem of picking salt boundaries.

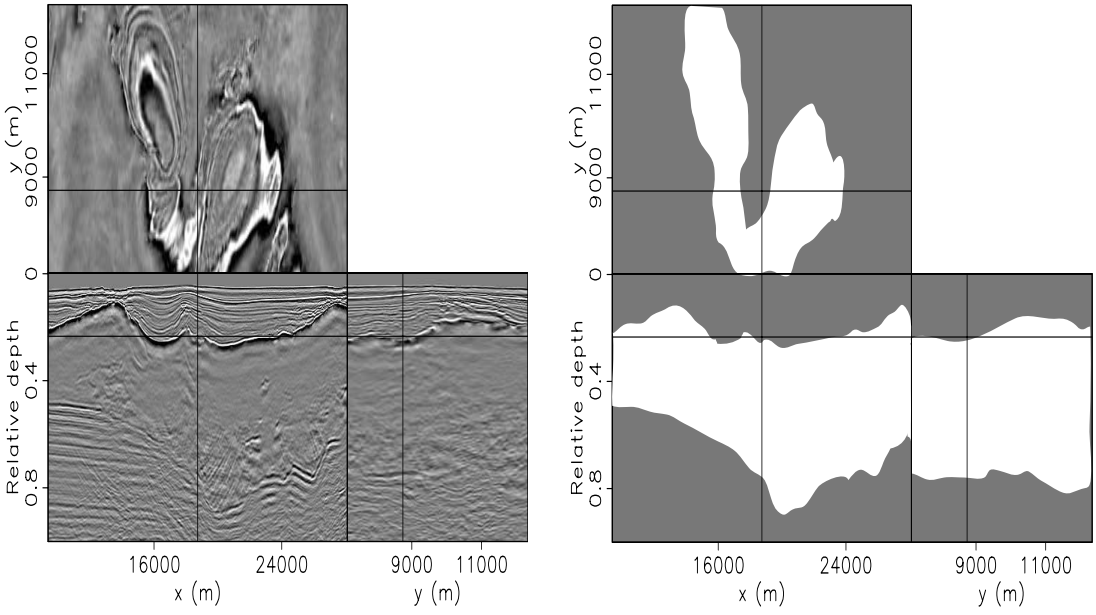

Figure 3.1: The voxels in the image on the left are clustered into the two groups on the right. The white group is inside the salt where as the gray group is outside the salt. The salt interface is where the two groups meet. segment-merge_cart [NR]

To track salt boundaries, this method segments seismic data images into two groups or clusters, one inside the salt and the other outside it. The picked salt interface is where the two groups meet. This is illustrated in Figure 3.1. To find the interface, weights are created relating each voxel to every other voxel within a neighborhood. The weights relating two voxels are made weak when the chosen attribute or attributes indicate a salt interface between them, and strong otherwise. For example, if instantaneous amplitude is the chosen attribute, then the weight will be weak if there is a bright amplitude boundary between the two voxels. Once the weights are found, an optimum partition is found by minimizing the normalized sum of

the weights cut by the partition. This optimum partition is found by solving an eigenvector problem.

Although there are many algorithms for image segmentation, I chose to use NCIS because it provides a globally optimized solution that is easy to adapt to the problem of seismic salt picking. The simplest kind of segmentation is pixel-based segmentation. This involves thresholding images according to pixel, or voxel, values. In the context of seismic data, this may simply be threshholding attributes. This approach may work well in places but behave erratically in other places. More consistent solutions can be found using globally optimized pixel-based segmentation algorithms (Hanning and Pisinger, 2002). Similarly, K-means clustering (MacQueen, 1967) could be used to partition an image into two groups by minimizing the difference between the value at each pixel and the centroid of a chosen attribute. Both globally optimized pixel-based segmentation and K-means methods should be able to locate a salt interface using attributes that have significantly different values inside the salt than outside the salt, but it is not obvious how to apply these methods using attributes like amplitude where the interface is defined by a bright event. Alternatively, the stochastic clusters method (Gdalyahu et al., 2001; Hale and Emanuel, 2003) iteratively combines adjacent pixels probabalistically until an image is sufficiently segmented. The probability between each pixel is essentially the same as the weight described above for NCIS. However, unlike NCIS, the stochastic clusters method will not necessarily break an image into just two groups.

Although Shi and Malik (2000) implement several cost-saving measures, such as sparse matrix storage and random sampling, their technique is still very costly for 3D seismic datasets. By imposing inner- and outer-bound constraints, I greatly reduce the effective problem size and thereby reduce the computational requirements and increase the precision. There are two significant computational time bottlenecks in this algorithm. The first bottleneck is the creation of the weight matrix which must relate every point to every other point in a local neighborhood. The second bottleneck is the estimation of the eigenvector, which requires numerous matrix-vector products involving the large weight matrix. I have distributed both of these cumbersome calculations on a parallel cluster, significantly improving the performance and practicality of this method.

In this chapter, I present a parallel implementation of the modified NCIS technique for

tracking 3D salt boundaries. I first review the segmentation methodology and describe how I have adapted it by using attributes to pick seismic salt boundaries. I also discuss how my method can provide alternative picking solutions where the correct solution is not obvious. I also describe how to introduce bound constraints and alleviate resulting artifacts with novel boundary conditions. Next, I illustrate the use of this method with bound constraints on a 2D field data set. I then describe how I have parallelized the calculation of both the weight matrix values and the matrix-vector products of the eigenvector calculation. Finally, I demonstrate the applicability of this technique on a 3D field seismic datacube.

SEGMENTATION METHODOLOGY

The NCIS technique partitions images into two groups. To do this, it first creates weights relating each sample to other samples within local neighborhoods, and stores them in a matrix **W**. It then finds the cut that partitions the image into two groups, A and B, by minimizing the normalized cut:

$$\mathrm{N_{cut}} = \frac{\mathrm{cut}}{\mathrm{total_A}} + \frac{\mathrm{cut}}{\mathrm{total_B}}, \tag{3.1}$$

where cut is the sum of the weights cut by the partition, $\mathrm{total_A}$ is the sum of all weights in Group A, and $\mathrm{total_B}$ is the sum of all weights in Group B. Normalizing the cut by the sum of all the weights in each group prevents the partition from selecting overly-small groups of pixels.

The objective of NCIS is to partition an image so that the normalized cut is minimized, but an exhaustive search of all possible partionings is prohibitively expensive for practical problem sizes. A binary partitioning vector **x** whose elements classify each pixel of an image into either Group A or B can be found via an eigenvector problem. Shi and Malik (2000) define a diagonal matrix **D** where each diagonal element is the sum of each column of the weight matrix **W**. They wish to find the indicator vector $\mathbf{x} \in \{1, 0\}$ that minimizes the normalized cut as

$$\min_{\mathbf{x}} \mathrm{N_{cut}}(\mathbf{x}). \tag{3.2}$$

They then show that this can be rewritten as a Rayleigh quotient (Golub and Loan, 1989)

$$\min_{\mathrm{y}} \frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}, \tag{3.3}$$

subject to the following two conditions:

$$\mathbf{y} \in \{1, -b\}, \tag{3.4}$$

$$\mathbf{y}^T\mathbf{D}\mathbf{1} = 0, \tag{3.5}$$

where $\mathbf{1}$ is a vector of ones. The first condition restricts the elements of $\mathbf{y}$ to one of two values (the value of b is not important to this discussion). The first conditon is relaxed so that $\mathbf{y}$ can take on real values (this has implications discussed later). In general, a Rayleigh quotient is minimized by the eigenvector with the smallest eigenvalue (Golub and Loan, 1989). Further, if the Rayleigh quotient is subject to constraints, it is minimzed by the eigenvector with the smallest eigenvalue that satisfies the constraints. Therefore the eigenvector with the smallest eigenvalue of the generalized eigensystem

$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y} \tag{3.6}$$

should minimize the unconstrained Rayleigh quotient in equation 3.3. However, in this case, the authors show that the eigenvector with the smallest eigenvalue is constant, $\mathbf{y} = \mathbf{1}$. This does not satisfy the second condition (equation 3.5): $\mathbf{1}^T\mathbf{D}\mathbf{1} \neq 0$.. Since the eigenvector with the second smallest eigenvalue is orthogonal to the first, the eigenvector with the second smallest eigenvalue does not violate the second condition and is, in fact, the eigenvector used to partition the image so that the normalized cut is minimized.

The eigenvector ($\mathbf{y}$) with the second smallest eigenvalue (λ) is used to partition the image by taking all values greater than zero to be in one group, and its complement to be in the other. However, splitting the eigenvector with the second smallest eigenvalue of equation 3.6 at zero only approximates the optimum normalized cut. It is not an exact solution because of the relaxing of the first condition (equation 3.4) to allow $\mathbf{y}$ to take on real values. As a result, Shi and Malik (2000) recommend recalculating the normalized cut in equation 3.1 for other

non-zero splitting points to find improved partitions.

For a more intuitive understanding, the NCIS algorithm has been linked to random walks by Meila and Shi (2001, 2000). They show that normalizing the weight matrix $\mathbf{W}$ yields a stochastic matrix

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}. \tag{3.7}$$

In the context of Markov random walks, the elements of a stochastic matrix represent the probability of moving from one pixel to another in one step. For example, the pixel corresponding to a particular row of the matrix will jump to any one of the other pixels in the image with the probablity specified by the corresponding column element. Meila and Shi (2001, 2000) also show that the eigenvectors of this stochastic matrix are identical to the eigenvectors of the generalized eigensystem in equation 3.6, with the corresponding eigenvalues reversed. Thus the largest eigenvalue of equation 3.6 corresponds to the lowest eigenvalue of the stochastic matrix. Therefore, taking the eigenvector with the second smallest eigenvalue of equation 3.6 is exactly the same as taking the eigenvector with the second largest eigenvalue of the stochastic matrix $\mathbf{P}$. Having linked Markov random walks and normalized cuts, Meila and Shi (2001, 2000) show that partitioning an image so that the normalized cut is minimized is the same as partitioning an image so that in the next step of the random walk, pixels will tend to stay in their respective groups.

The NCIS method can be very computationally expensive. If the image being segmented has N samples, its corresponding weight matrix has N^2 elements. Furthermore, weights are created between each pixel and all other pixels within a circular neighborhood in 2D and a spherical neighborhood in 3D. This can result in a large number of non-zero weights.

Shi and Malik (2000) describe two measures to substantially mitigate these computational challenges. The first is to store only non-zero values of the weight matrix. Because the pixels are related to other pixels within a limited search window, the weight matrix tends to be full of zeros. By using sparse matrix storage, only a fraction of the memory is required. The second measure is random sampling. Rather than creating weights between each pixel and every other pixel within neighborhoods, pixels are sampled randomly reducing computational cost and memory requirements by as much as 90 percent. This is illustrated in the cartoon in

Figure 3.2. The advantage of random sampling over other sampling alternatives is that it is still relatively unbiased in direction and distance. For example, the same number of points are sampled regardless of distance from the center.

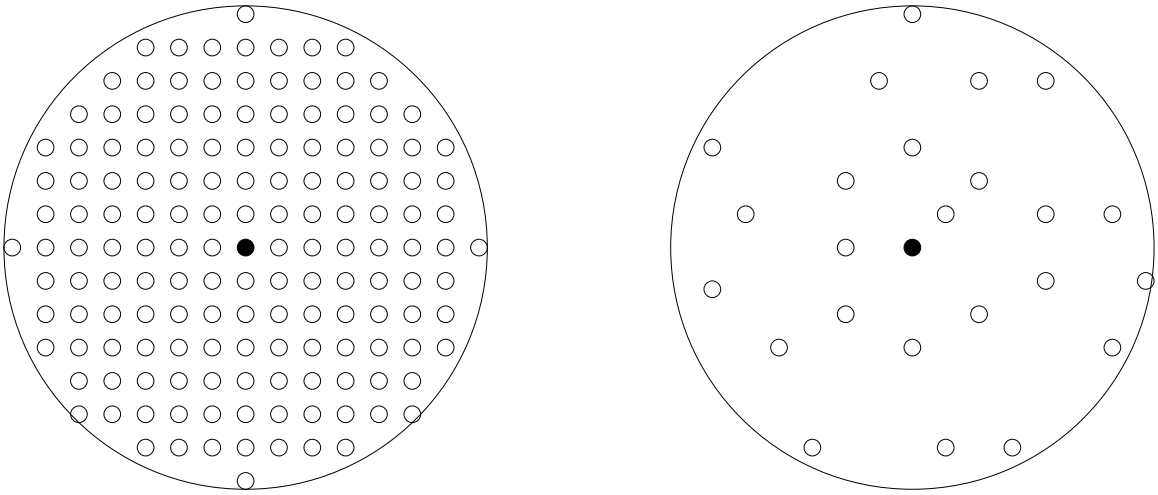

Figure 3.2: A cartoon illustrating dense sampling (left) and random sampling (right). In each case the black pixel in center is weighted to every other pixel in the local neighborhood, defined by the large circle. Random sampling significantly reduces the number of weights calculated without introducing significant biases in direction or distance. segment-sample [NR]

APPLICATION TO SEISMIC

To apply this segmentation method to seismic data, I first modify the weight calculation to handle seismic attributes. Next, I implement bound constraints. Throughout this discussion, I illustrate the process on 2D synthetic models and cartoon figures.

I want to modify the weight calculation so that weights connecting pixels on either side of the salt interface are low and the weights connecting pixels on the same side of the salt interface are relatively high. Thus, I need to find an attribute or combination of attributes that contains information on the location of the salt interface.

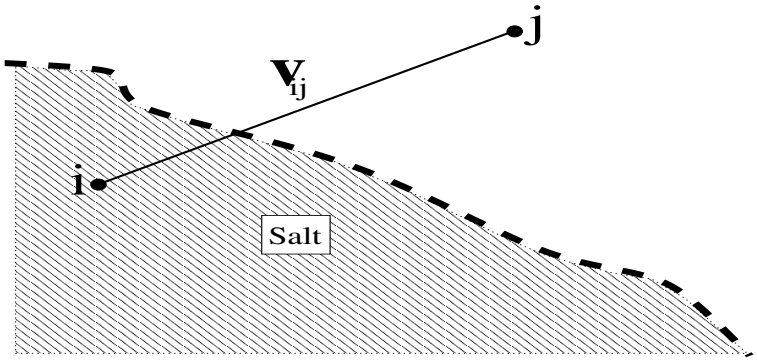

Figure 3.3: The weight between pixels i and j is dependent upon an attribute that identifies the dashed boundary or an attribute that identifies the regions of the salt (cross-hatched) versus the adjacent sediments. $\mathbf{v}_{ij}$ is a vector representing the shortest path between the two pixels. segment-fig5 [NR]

Building the weight matrix

I have identified two different attribute classes that must be treated differently to build the weight matrix. Attributes identify the salt interface either as a peak (as in the case with amplitude identifying a bright amplitude salt reflector) or as the border between two regions. In the former case, the weight is measured by taking the maximum value of the envelope of the complex trace (instantaneous) amplitude sampled along the shortest path between two pixels. In the latter case, the weight is measured by differencing the values at the two pixels. Both methods are illustrated in Figure 3.3. Let $\mathbf{v}_{ij}$ be a vector representing the shortest path between two pixels i and j but excluding the pixels themselves. If the attribute value at the salt interface (dashed) is a local maximum value, the weight connecting two pixels i and j is determined from the maximum of the normalized attribute A sampled along $\mathbf{v}_{ij}$ and a distance weight term as follows:

$$W_{ij} = \begin{cases} \exp(\frac{-\|\max A(\mathbf{v}_{ij})\|^2}{{\epsilon_A}^2}) \exp(\frac{-\|X_{(i)} - X_{(j)}\|^2}{{\epsilon_X}^2}) & \text{if min A}(\mathbf{v}_{ij}), \\ 1 & \text{otherwise.} \end{cases} \tag{3.8}$$

The values ϵ_A and ϵ_X are user-specified tolerances that control the sensitivity of the weight to attribute A and distance X, respectively.

The weights are calculated only within the local neighborhood, beyond which they are assumed to be zero (Shi and Malik, 2000). Of course, if ϵ_X is small enough to cause the weight to drop below the sparse threshold (smallest weight value to be stored) then it effectively controls the size of the local neighborhood. The size of this neighborhood is based on competing requirements of computational efficiency and accuracy of results. Typically, larger neighborhoods result in more accuracy because more pixels on the same side of the salt interface are weighted to each other. This, of course, requires storing and manipulating more non-zero values.

If the salt interface is determined as the border of clusters of pixels of similar attribute intensity, the weight is dependent only on the absolute value of the attribute and is therefore resolved by the attribute differences at the two pixels and a distance term as follows:

$$W_{ij} = \exp(\frac{-\|F_{(i)}-F_{(j)}\|^2}{{\epsilon_F}^2})\exp(\frac{-\|X_{(i)}-X_{(j)}\|^2}{{\epsilon_X}^2}), \tag{3.9}$$

where ϵ_F is a user specified tolerance that controls the sensitivity of the weight to attribute F.

There are situations where a binary weight is more robust. In this case, the weights will be zero if the attribute is above a user-specified tolerance, and one otherwise. This would be appropriate when using instaneous amplitude with noisy data. Additionally, this has significant computational savings by requiring only binary storage.

Multiple attributes can be easily combined by multiplying their respective weights. In the case of two attributes, one using a local boundary salt measure and the other using two regions, the combined weight is

$$W_{ij} = \begin{cases} \exp(\frac{-\|\max A(\mathbf{v}_{ij})\|^2}{{\epsilon_A}^2})\exp(\frac{-\|F_{(i)}-F_{(j)}\|^2}{{\epsilon_F}^2})\exp(\frac{-\|X_{(i)}-X_{(j)}\|^2}{{\epsilon_X}^2} & \text{if min A}(\mathbf{v}_{ij})), \\ 1 & \text{otherwise.} \end{cases} \tag{3.10}$$

Alternatively, if both attributes identify the salt interface with a peak, I can first multiply the attributes and treat them as a single attribute for weight generation.

The values of parameters ϵ_A, ϵ_X, ϵ_F, the number of random samples per pixel, and the

radius of search neighborhood should first be determined on a small subset of the data. Unfortunately, if the full data set is significantly different than the subset, results can be inaccurate. In such cases, it may be necessary to break the data set into pieces, segment individually, and then combine the results.

The choice of attribute or combination of attributes is problem dependent, and may change within a volume, in which case, it is best to segment individual pieces. In all cases, care should be taken to choose attributes that are not very sensitive to the velocity model used for migration. Otherwise, the abrupt velocity contrast at the salt interface may strongly influence the picking result. Furthermore, adding more attributes adds more parameters to adjust, increasing the complexity of finding optimal parameter values.

An important practical caveat concerns pixels located on the salt interface. The simple weight criteria just described will give these pixels weak weights relative to all other pixels and will tend to cluster them into their own group. To avoid this, I add the requirement that salt interface pixels can be weak only in one direction.

Gaps, multiple paths, and uncertainty

The eigenvector provides valuable information beyond simply identifying the salt interface. It also contains information about the certainty of its picking result and, in some cases, alternative picking paths. For instance, in places where the salt interface is well defined by the attributes, the weight will be very steep and abrupt, with a profile much like a step function. Where the interface is not well defined, the eigenvector will be smooth and sloping. Where the attributes define multiple, equally probable paths, the eigenvector can have multiple steps, each identifying a different possible path. This is illustrated by the following example.

Figure 3.4a shows a simple 2D synthetic model of a discontinuous interface. Using the weight criterion in equation 3.8, the resulting eigenvector is displayed in Figure 3.4b. The final step of the method is to use the zero contour of this eigenvector as the salt interface. The uncertainty in the vicinity of the gap can be observed in the smoothness of the eigenvector. This can also be observed in the contours of the eigenvector displayed in Figure 3.4c. Note that the presence of the gap affects the certainty even away from the gap itself. This is a

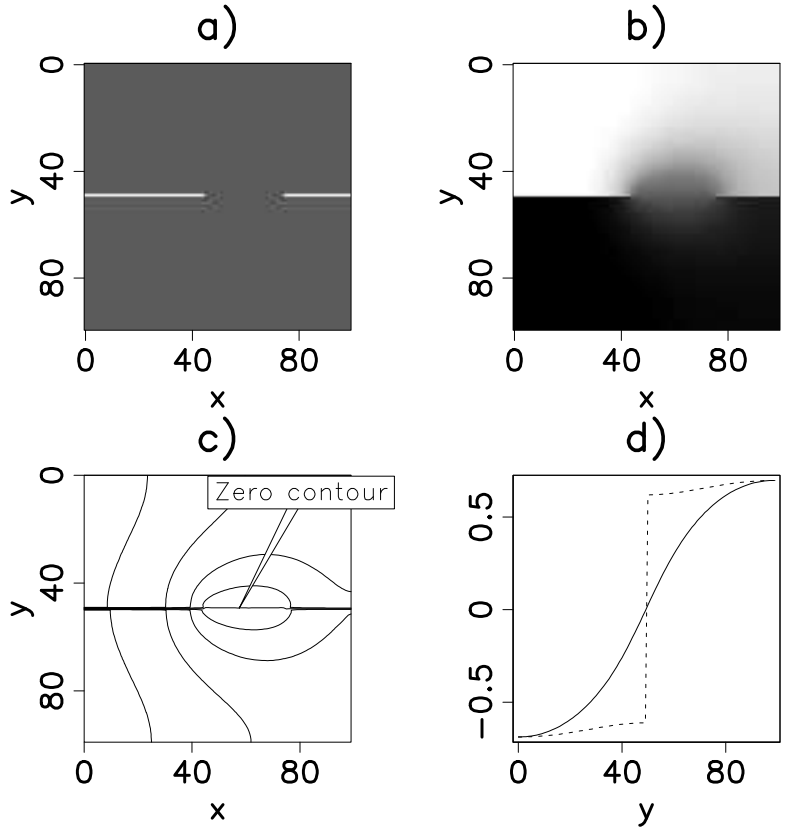

Figure 3.4: (a) Synthetic model with a discontinuous interface. (b) The eigenvector with the second smallest eigenvalue estimated using (a). (c) Contoured (b). (d) Plot of the normalized eigenvector with the second smallest eigenvalue in (b) at x=20 (dashed) and x=60 (solid). Notice the relationship of steepness to certainty. segment-mod2.combo [CR]

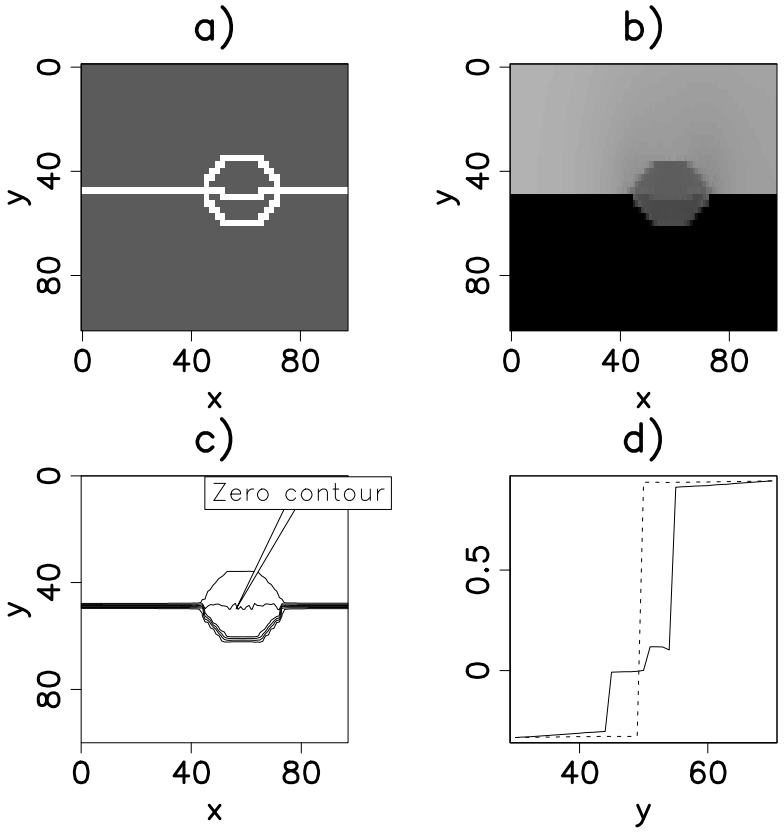

Figure 3.5: (a) Synthetic model with an interface with multiple solutions. (b) The eigenvector with the second smallest eigenvalue estimated using (a). (c) Contoured (b). (d) Plot of the normalized the eigenvector with the second smallest eigenvalue in (b) at x=20 (dashed) and x=60 (solid). The shape of all three solutions is captured in the eigenvector. segment-mod5.combo [CR]

consequence of the global nature of this segmentation method. The zero contour goes straight across the gap and, in this case, is the desired solution.

Figure 3.4d shows two graphs extracted from the eigenvector in Figure 3.4b. The dashed line is taken from x=20, where the step function shape illustrates that the interface is well defined by the attribute. The solid line is taken from x=60, right through the center of the gap. Its shape is smooth, illustrating the uncertainty. In general, the zero contour of the eigenvector is an adequate splitting point to partition the image. However, Shi and Malik (2000) also recommend trying other solutions from non-zero contours.

In Figure 3.5a is a synthetic with three possible salt boundaries. This model represents interpretive situations where multiple solutions are present and can be settled only by manual interpretation. The resulting eigenvector in Figure 3.5b has four regions illustrated by different shades of gray. The contours in Figure 3.5c show that there are three possible solutions encapsulated within the eigenvector and the zero contour is just one possible solution. This multivalued result is also evidenced in the graph in Figure 3.5d. An interpreter could easily select either of the two alternative paths by selecting a different contour (splitting point). Hence, the eigenvector itself has interpretation potential built into it.

MODIFICATIONS FOR EFFICIENCY

Although sparse matrix storage and random sampling already significantly reduce the computational requirements of the method, to apply it to 3D seismic data cubes additional measures are required. Here I describe two steps that greatly alleviate the computational requirements of this method. The first is the introduction of bound-constraints. The second is the distribution of two key steps of this algorithm on a parallel cluster.

Random links at boundaries

Although the weight matrix (**W**) is stored as a sparse matrix (only non-zero values are stored), it can still be very large, the size of the input image multiplied by the number of samples taken from each pixel's search neighborhood. In 3D, the sparse matrix can be 120 to 300 times the

size of the input image. Therefore, any reduction in the size of the input image is helpful. By limiting the region where an interface can exist, applying bound constraints greatly reduces the size of the problem. These bounds can be acquired from several sources, for instance, by first running the algorithm with small search neighborhoods and coarse sampling, or by using a coarse initial interpretation.

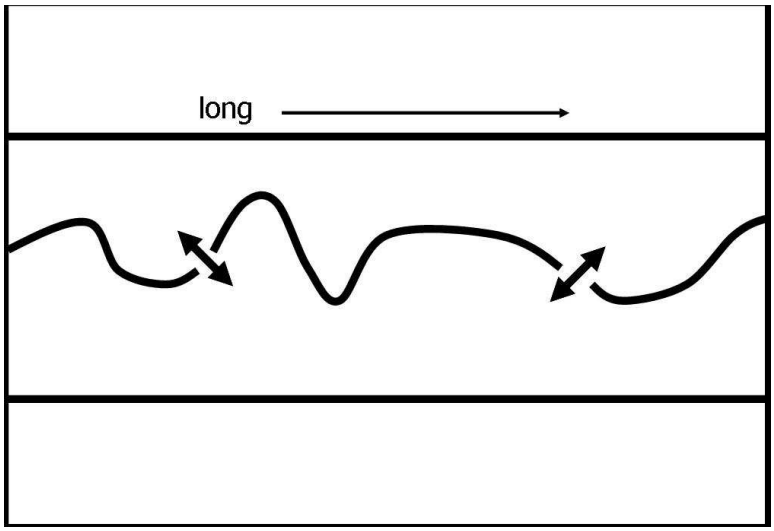

Figure 3.6: A cartoon of a masked salt interface. The image is long and thin with a discontinuous salt interface snaking across it. segment-pic2 [NR]

Unfortunately, the NCIS method tends to partition elongated images along their shortest dimension. For instance, Figure 3.6 is a cartoon of an elongated image with a salt interface snaking across it. If the segmentation algorithm were to function as hoped, the minimum cut would be found along the salt interface. However, because the salt interface is discontinuous, it is likely that the minimum of the normalized cut in equation 3.1 will be found by cutting the image vertically where the image is thin. A discontinuous interface has places where the interface is not well defined. The locations are indicated by arrows in Figure 3.6. To correct this problem, I exploit the fact that the upper boundary will necessarily be in Group A and the lower boundary will be in Group B. In other words, I want to force the segmentation algorithm to put the coarsely picked bounds in different groups.

I can enforce this constraint during the creation of the weight matrix ($\mathbf{W}$). Recall that this

matrix contains weights relating each sample to other samples along paths within a neighborhood. For any given sample, if its search neighborhood happens to cross a coarsely picked boundary, it becomes weighted to another sample at a random distance along the boundary. This can be imagined by wrapping the image on a globe so that both the upper and lower bounds collapse to points at the poles. When estimating the weight matrix, every time a path crosses the north or south pole, it continues down the other side. The distance weight of equations 3.8-3.10 is now the distance along the path rather than the absolute distance. By implementing these "random" boundaries, I am effectively removing the upper and lower boundaries.

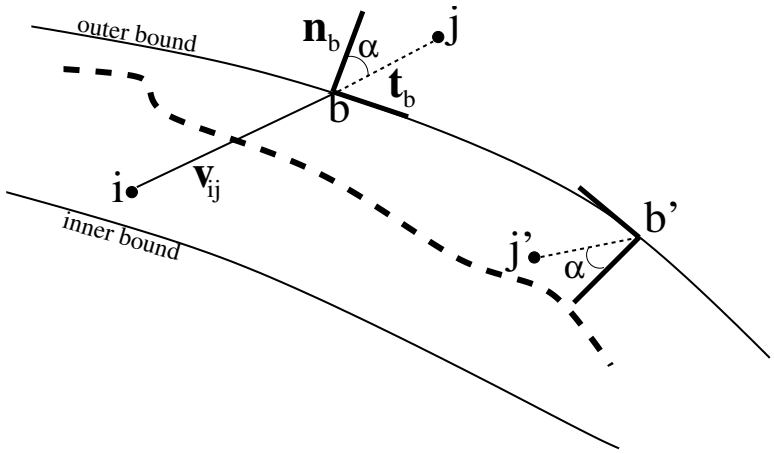

Figure 3.7: A cartoon illustrating random bounds. The weight between pixels i and j along vector $\mathbf{v}_{ij}$ is altered to $\mathbf{v}_{ij'}$, because j crosses the outer boundary. Whenever a path crosses an inner or outer bound constraint, it is reflected back into the bounded region at a random location along the bound. segment-fig3 [NR]

In Figure 3.7, two pixels, i and j, of a 2D image are plotted separated by a boundary indicating that pixel j is outside of the bounds. Vector $\mathbf{v}_{ij}$ represents the shortest path connecting them. $\mathbf{n}_b$ and $\mathbf{t}_b$ are the unit normal and tangent vectors where $\mathbf{v}_{ij}$ crosses the boundary at location b. Together, $\mathbf{n}_b$ and $\mathbf{t}_b$ make a basis onto which vector $\mathbf{v}_{bj}$ is projected and then mapped to another basis at $\mathbf{b}'$, a random distance along the boundary. In summary, if j is outside the boundary, then $\mathbf{v}_{ij}$ is the sum of two segments:

$$\mathbf{v}_{ij} = \mathbf{v}_{ib} + \mathbf{v}_{bj}. \tag{3.11}$$

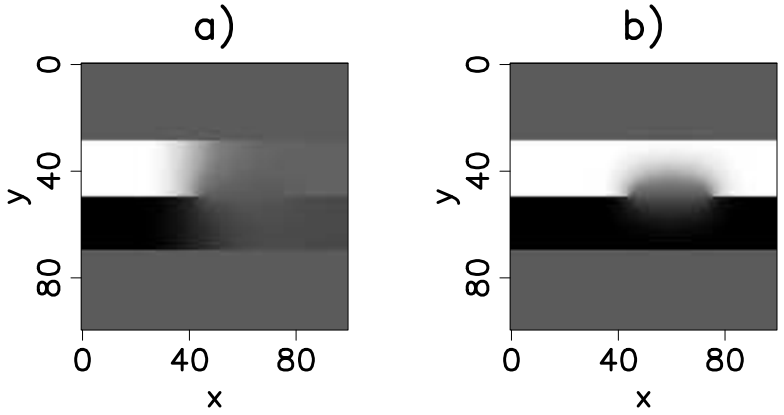

Figure 3.8: (a) The eigenvector with the second smallest eigenvalue estimated using the synthetic model in Figure 3.4a within a bounded region. (b) Same as (a) except using random links at boundaries. segment-mod6.combo [CR]

Segment $\mathbf{v}_{bj}$ is projected onto the basis vectors $\mathbf{n}_b$ and $\mathbf{t}_b$ then mapped to $\mathbf{v}_{b'j'}$ as:

$$\mathbf{v}_{b'j'} = \mathbf{b}' - (\mathbf{v}_{bj} \cdot \mathbf{n}_b)\mathbf{n}_{b'} - (\mathbf{v}_{bj} \cdot \mathbf{t}_b)\mathbf{t}_{b'}. \tag{3.12}$$

The new randomly linked path is

$$\mathbf{v}_{ij'} = \mathbf{v}_{ib} + \mathbf{v}_{b'j'}. \tag{3.13}$$

Figure 3.8a shows the resulting eigenvector used for segmentation when applied to the same 2D synthetic model shown Figure 3.4a except within a bounded region. The segmentation is now restricted to a narrow portion of the image. In this case, the problem size is reduced by about 60 percent. It is clear that this result will be difficult to split into two regions along the desired interface. Figure 3.8b is the eigenvector that results using random bounds described in the previous section. Notice that it is very similar to the unconstrained result in 3.4b and would be easy to split along the desired interface.

2D FIELD TEST CASE

I tested this method on a 2D migrated section taken from a Gulf of Mexico data set provided by WesternGeco. Figure 3.9 shows a base of salt reflection. Although it is discontinuous, the human eye can see how it should be picked without too much uncertainty. In this example, I use only the instantaneous amplitude attribute for segmentation.

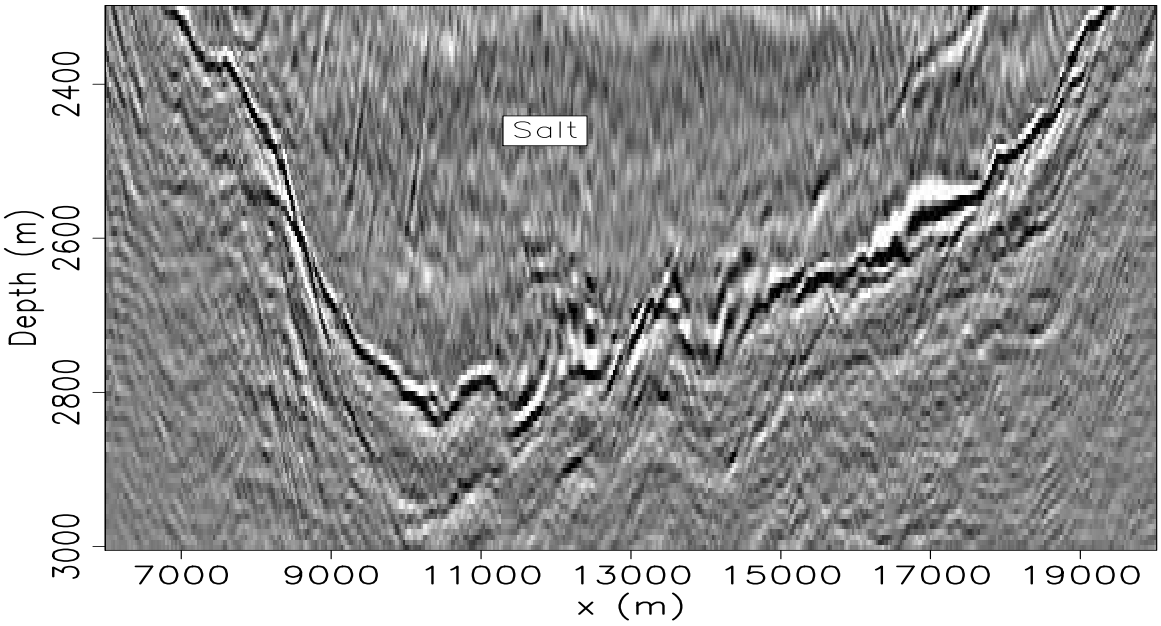

Figure 3.9: A 2D seismic section with a salt interface from the Gulf of Mexico. segment-gom2D [CR]

I first ran the standard segmentation method on the entire image in Figure 3.9 with coarse sampling, using every 4th sample in the depth and every 12th sample in the x direction. It was necessary to pad the input amplitude with zeros to make the image square and prevent segmenting along the lower boundary. I then widened the resulting boundary by a radius of 15 samples to define a bounded region shown in Figure 3.10.

I then ran the standard segmentation method on the masked area without applying the random links at the boundaries. The resulting eigenvector ($\mathbf{y}$) used to partition the image is shown in Figure 3.11. In principle, this eigenvector would be clipped to track the salt. Unfortunately, it is dominated by a low frequency trend and will create an erroneous vertical

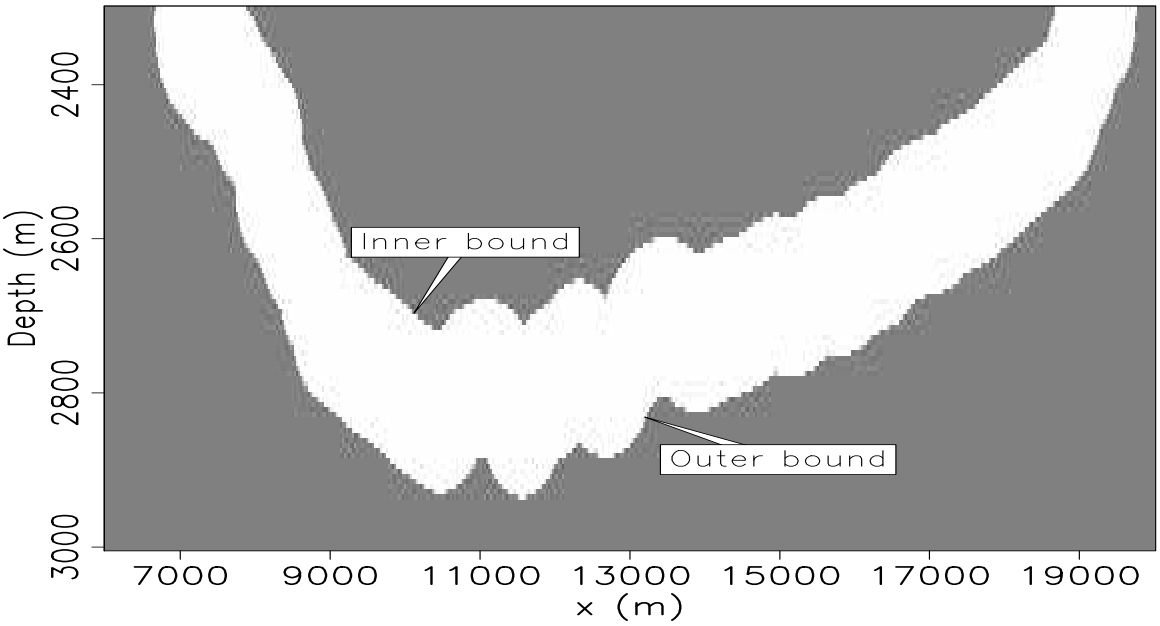

Figure 3.10: A binary mask defining the bounds of the salt interface. This was generated by first running the image segmentation method on the entire image with short search distances and sparse sampling. segment-gom2D_mask [CR]

cut.

The eigenvector result of applying the method with random links at boundaries is shown in Figure 3.12. This was generated using ϵ_A=.25, ϵ_X=8 (I found in this case that the distance term of equation 3.8 was unimportant), and a local neighorhood radius of 30 samples. It is clear that the salt interface can be extracted from this image. Notice that the eigenvector is smooth (or gray) in areas of greater uncertainty. Where the interface is clearly defined, the eigenvector is steep. The upper left corner appears to be smooth, because of a boundary effect, where random links are not connected across the top.

The gray areas in Figure 3.12 indicate that the eigenvector has uncertainty that can be exploited as alternative picking solutions. In Figure 3.13 are the picking results using both zero (solid) and non-zero (dotted) spliting points. For the non-zero result, I normalize the eigenvector to range from -1.0 to 1.0 and then split it at -0.5. Notice that in areas of low uncertainty (where the eigenvector is steep), the solution does not change between the different

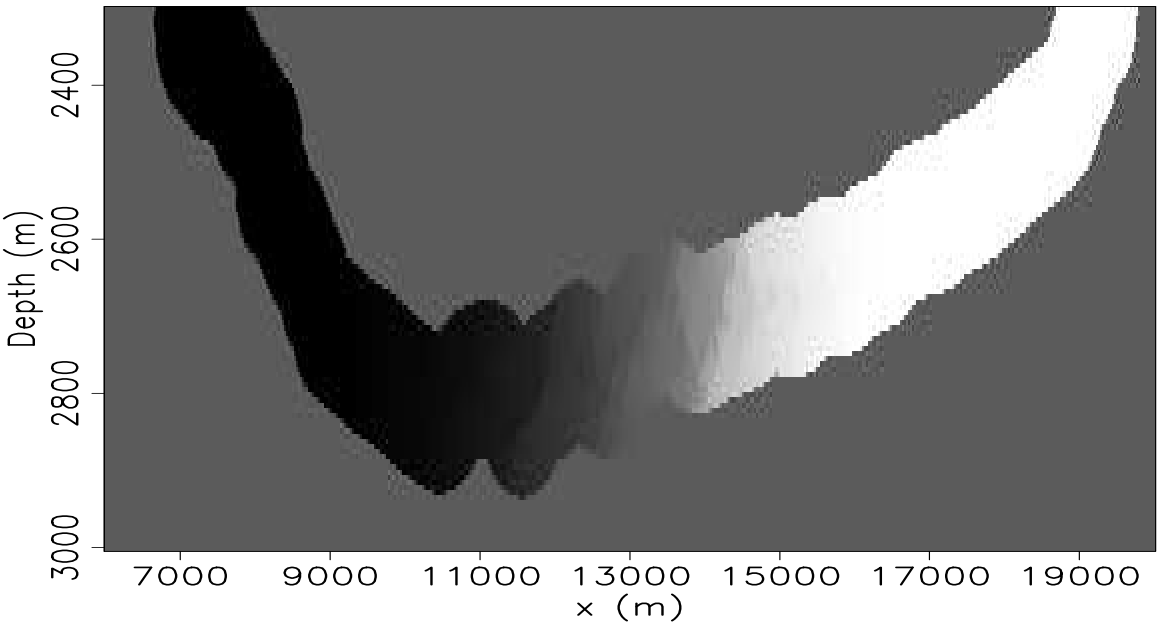

Figure 3.11: The eigenvector that is used to partition the image using the old segmentation algorithm after the mask in Figure 3.10 is applied. Hints of the salt interface can be seen but is obscured by the overall low frequency trend from left to right. This will not produce a satisfactory segmentation result. segment-gom2D_nb.eig [CR]

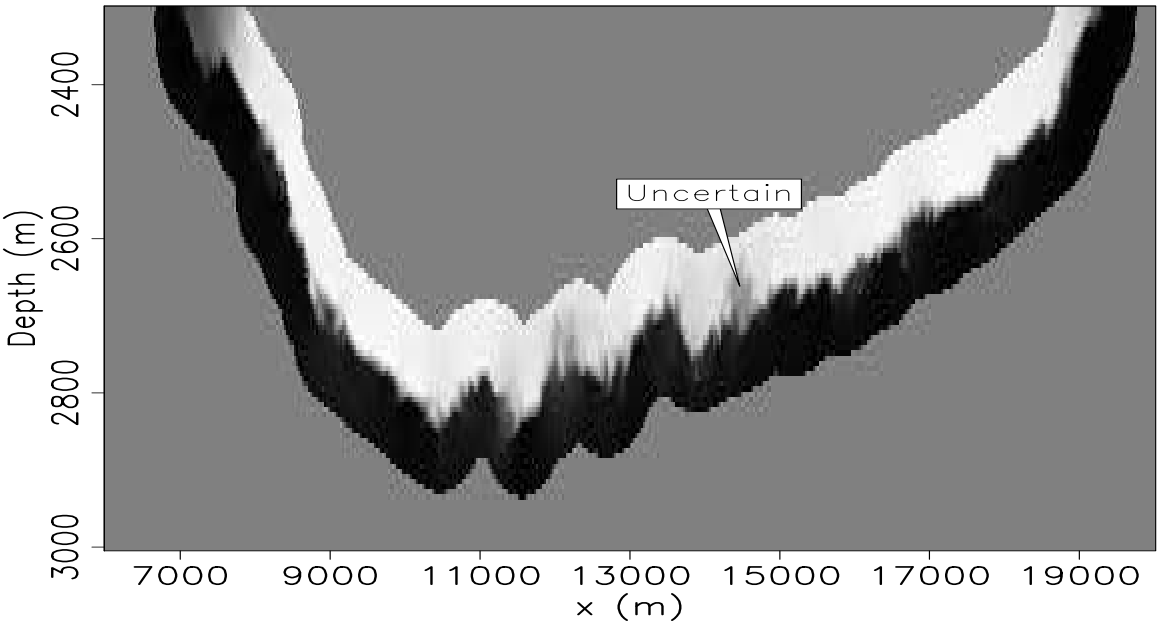

Figure 3.12: The eigenvector that is used to partition the image using the approach with random bounds. The salt interface is clear in this image and can be extracted easily. segment-gom2D.eig [CR]

splitting points. As expected, in other areas, the solution changes significantly.

Overall, this method does an excellent job of tracking the salt interface. Because the weights are created from instantaneous amplitude, the phase of the pick is not constant. This is an important caveat of this method. For tracking a particular phase of an event, the computation of the weights should be modified.

3D IMPLEMENTATION

The most significant difference between 3D image segmentation and 2D image segmentation is the generation of the weight matrix; the rest of the algorithm is almost identical. When creating the weight matrix, instead of randomly sampling from a circular neighborhood, I sample from a sphere. Of course this means that more points are sampled per voxel, so the sparse matrix has considerably more nonzero values, and the entire algorithm is more expensive.

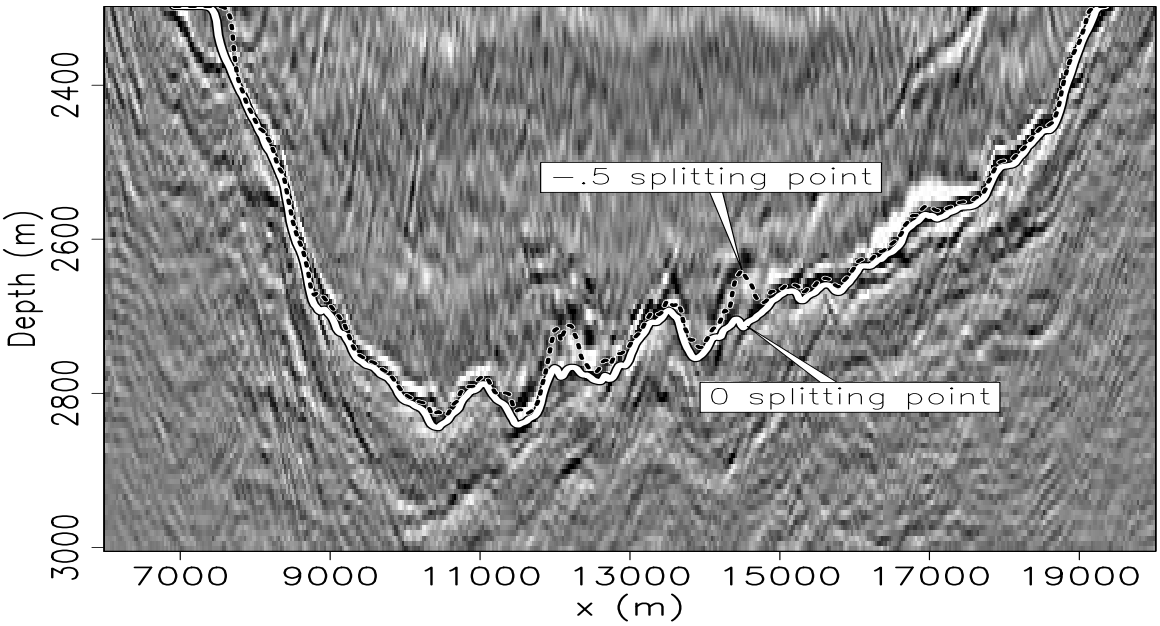

Figure 3.13: The interface is garnered from the image in Figure 3.12 using two different splitting points and overlain on the original base of salt data. The solid white line is using the standard zero value splitting point. The dotted picking is the result of using a non-zero splitting point. segment-gom2D.horiz3 [CR]

Therefore, even with sparse matrices and tight boundaries, we still need to look for ways of reducing the computational cost of this algorithm for 3D problems.

PARALLEL IMPLEMENTATION

I identified two steps of this algorithm that could benefit from parallel implementation. The first step is the calculation of the weight matrix, where a parallel cluster can be used to significantly increase computation speed. The second is the estimation of the eigenvector with the second smallest eigenvalue, where parallel clusters can hold the entire sparse matrix in memory rather than storing and retrieving sections from disk.

Parallel calculation of the weight matrix

I have distributed the calculation of the weight matrix on a Beowulf cluster using the parallel infrastructure described in Clapp (2005). The complete image and boundary mask are distributed to each node. This is necessary because the random bounds require random jumps around the image. Different jobs are assigned different rows of the weight matrix, and no communication between nodes is necessary while calculating the weight matrix. Upon completion, the weights are collected into a single sparse matrix on the master node.

This parallel distribution is illustrated in the example in Figure 3.14 using the same synthetic used in previous examples. Copies of the entire image are sent to each node in the network. Each node is assigned a group of pixels from which to estimate weights, identified by the dashed grid in the figure. Similarly, each group of pixels corresponds to a group of rows in the weight matrix.

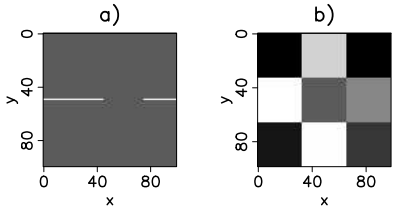

Figure 3.14: The same synthetic model as Figure 3.4a with overlaying dashed grid identifying groups of pixels assigned to individual nodes. segment-mod2grid.combo [ER]

Parallel calculation of the eigenvector

To calculate the eigenvector with the second smallest eigenvalue, I use ARPACK Fortran77 software (Lehoucq and Scott, 1996), as recommended by Shi and Malik (2000). This is a package of routines designed specifically for computing a few eigenvectors and eigenvalues for large sparse matricies.

The ARPACK interface requires the user to supply the subroutine that does the matrix-vector multiplication; which, being the most expensive portion of calculating the eigenvector, is the obvious target for parallelization. I implement the eigenvector calculation in a modified master-slave scheme. It is a systolic scheme where each slave node alternates between passing input/outputs and computing.

The slave nodes are intialized with a portion of the off-diagonal elements of the matrix. The master node is given a vector by the ARPACK library. It sends that vector to the first slave node and then begins to create the output vector by multiplying the diagonal terms of the matrix. Upon receiving the input vector, the first slave node passes the vector to the second slave node and then begins multiplying the input vector by its portion of the off-diagonal terms, creating its own output vector. This process is repeated by all of the slave nodes. The master node, upon finishing multiplying the diagonal terms, passes the output vector to the first slave node. This node then adds it to its output vector, and passes it to the second slave node. The process is repeated until the last slave node, which passes the completed matrix-vector multiplication back to the master node.

This scheme achieves a good level of load balancing with minimal communication wait time. For a seismic cube of $350 \times 500 \times 130$ samples, I ran the approach on a Infinband network of 8 nodes (Operton 875 processors with 2 gigibytes of memory each) and was able to do 200 iterations of 2 billion non-zero matrix elements in about 55 minutes.

The efficiency of this parallelized matrix-vector product depends on the speed of the nework. For most networks, the optimal number of nodes is what ever is the smallest number of nodes that can hold the sparse matrix in-core. The most efficient approach would be to store the entire sparse matrix in a shared memory OpenMP model.

3D FIELD TEST CASE

I tested the method on a Gulf of Mexico 3D data set provided by Unocal. This data is 350 X 500 X 130 samples and takes about an hour to run in parallel on 8 nodes (Operton 875 processors with 2 gigibytes of memory each). The upper salt interface is generally well defined, but there are several places that are not well defined (the velocity model could use some refinement); in those places, this dataset provides a significant picking challenge. This segmentation method can provide a quick, robust solution in the well defined areas, and a good guess (or guesses) elsewhere that is easily manually adjustable. The use of instantaneous amplitude alone provides an adequate solution in most areas, but to tackle the challenging areas I found a combination of amplitude and dip variability produces the best results.

The 3D seismic cube (Figure 3.15) has a prominent salt interface. The upper salt interface is well defined in most places but could benefit from some refinement of the velocity model. The lower interface is not so well defined but could also benefit from the improvements in the upper interface. In this case, I used a manually picked velocity model to define the upper and lower bounds. The width of the mask is approximately forty samples. Figure 3.16 shows the mask used to define the bound constraints. Alternatively, I could have garnered the bounds from a first pass of segmentation on a sub-sampled cube.

The eigenvector with the second smallest eigenvalue from the method using only instantaneous amplitude is displayed in Figure 3.17. This result was calculated using the weight criterion in equation 3.8 with $\epsilon_A = .02$ and $\epsilon_X = 8.0$. The value of ϵ_A was easily found by stepping in orders of magnitude. Since the quality of the results were insensitive to the distance weight, I effectively ignored the distance weight by using a relatively large ϵ_X. The local neighborhood radius was 20 samples and the number of randomly sampled points per node was 188. I recommend initially testing on a small window of data to find the best parameter values.

Three different orthogonal sections of a 3D cube are displayed. In most of the displayed sections, the eigenvector is steep, and its uncertainty is therefore low. However, at relative depth=0.2 and x=16000 m., the eigenvector appears to have multiple solutions. This is picking up the corresponding bright reflector that can be seen in Figure 3.15. Figure 3.18 displays the

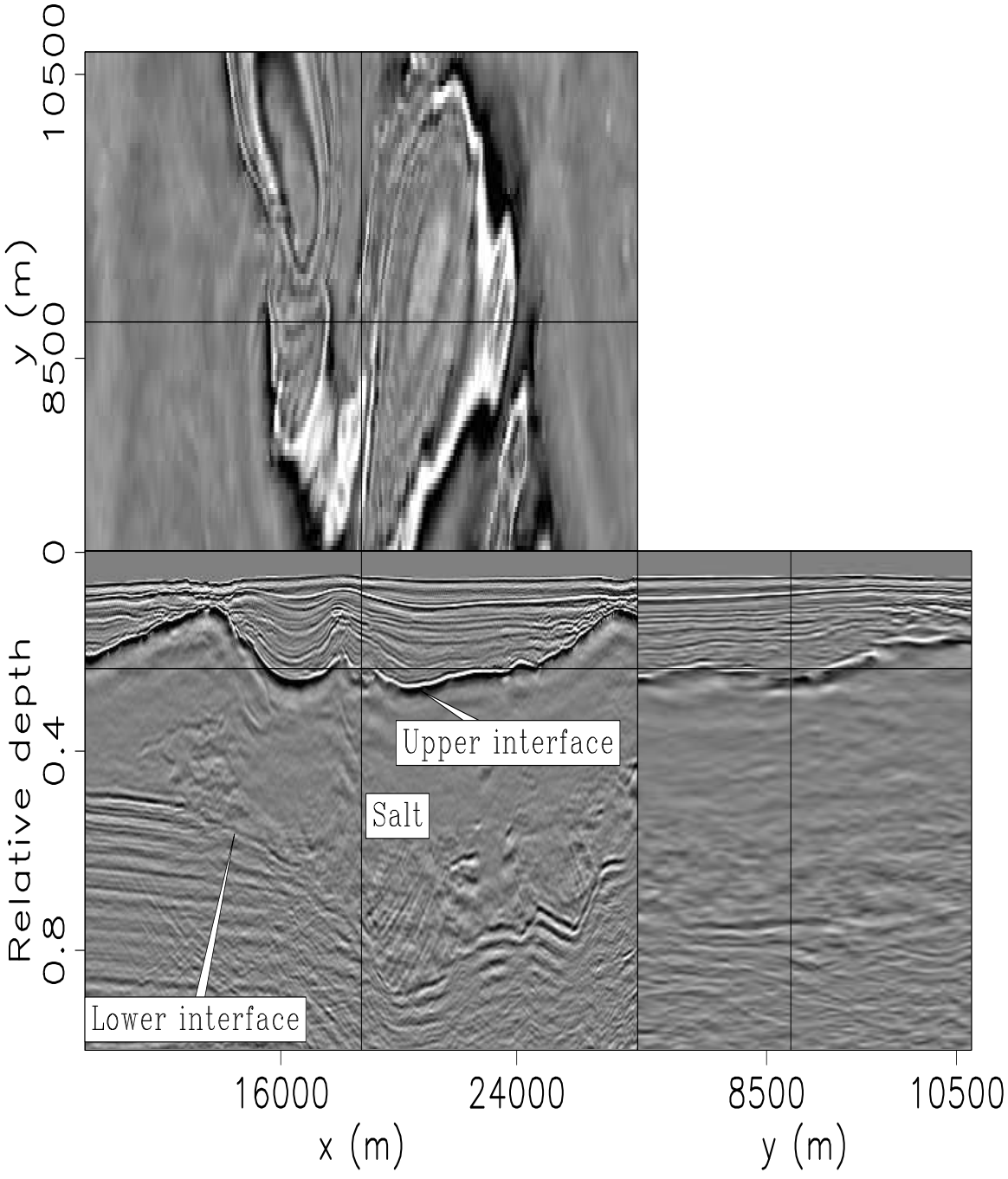

Figure 3.15: A Gulf of Mexico 3D data set with a prominent salt interface. Relative depth is used instead of absolute depth at the request of Unocal. The black lines superimposed onto the orthogonal sections identify the location of these sections: a depth slice at relative depth=0.235 , an in-line section at y=8760 m, and a cross-line section at x=18750 m. segment-dat_bw [CR]

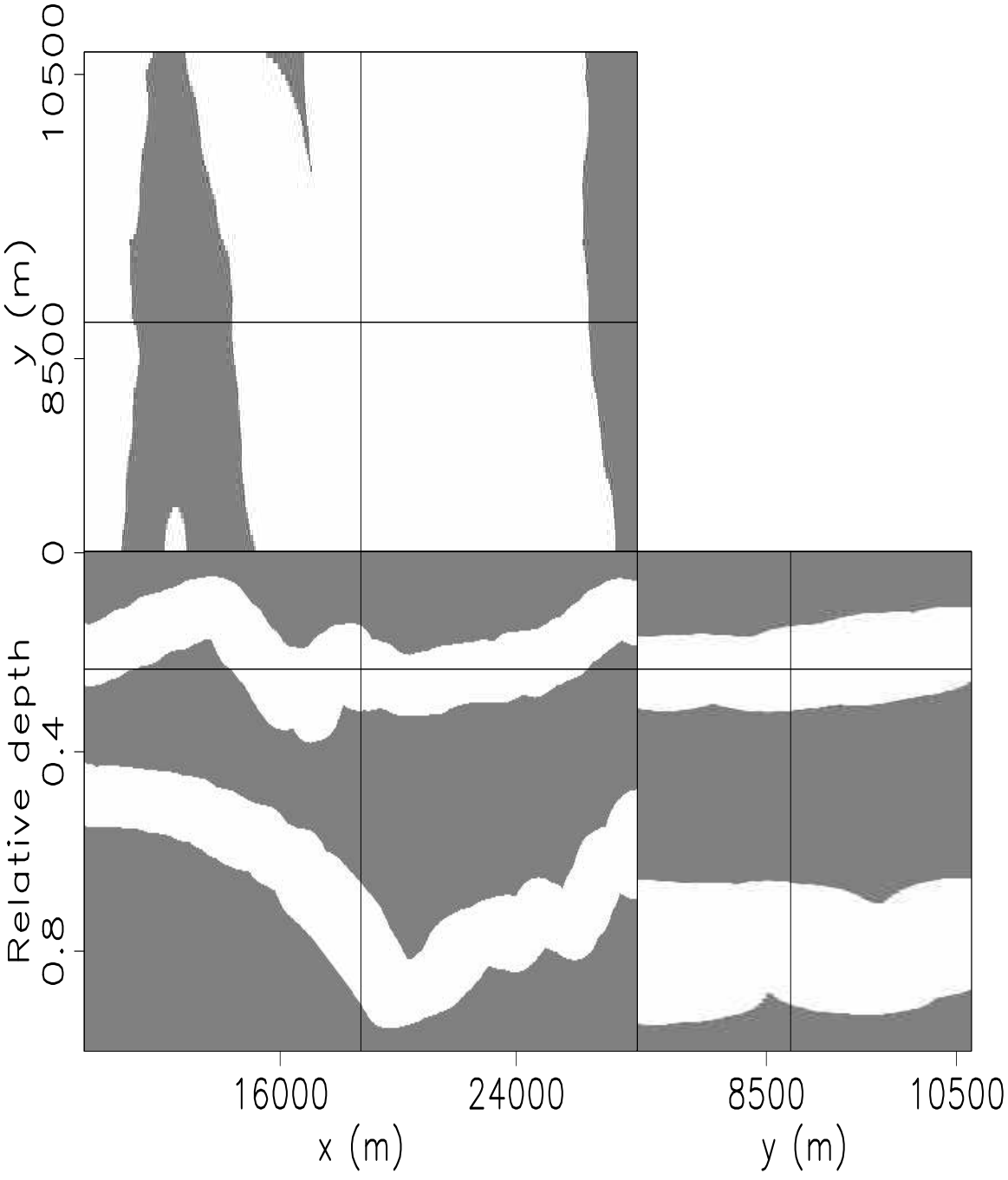

Figure 3.16: The mask used to define the upper and lower bounds of the salt interface of the data in Figure 3.15. This was created from the velocity model and is forty samples thick. This greatly reduces the size of the problem. In cases where a velocity model is not available, a similar mask can be created from segmenting a sub-sampled cube or from a rough manual interpretation. segment-mask_new [CR]

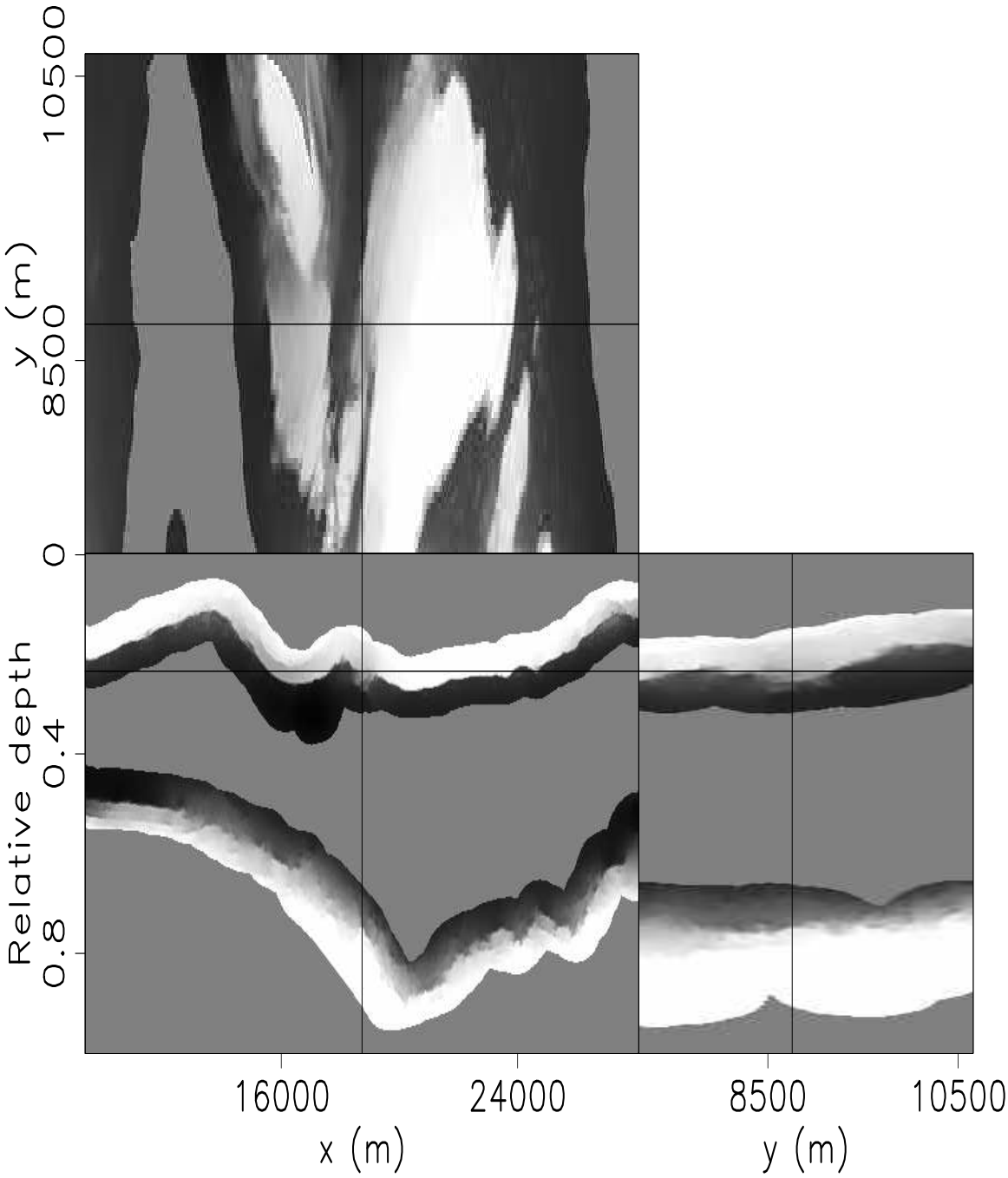

Figure 3.17: The eigenvector that is used to partition the image in Figure 3.15. Notice it is smooth where the amplitude does not delineate the interface well. segment-amp_eig1_new [CR]

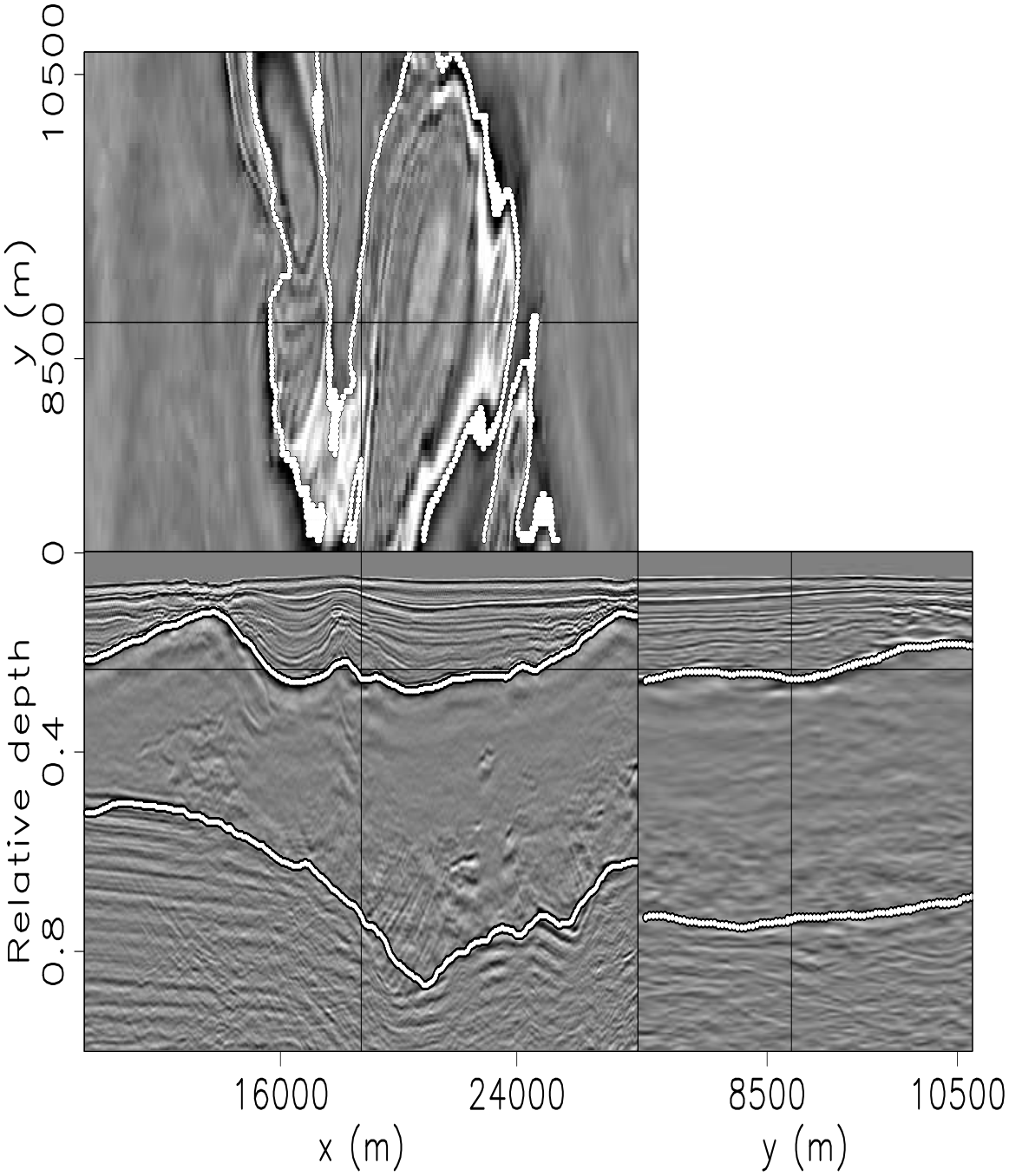

Figure 3.18: The interface is extracted from the image in Figure 3.17 and overlain on the image in Figure 3.15. In general, it accurately tracks the interface. segment-amp_pck1_bw [CR]

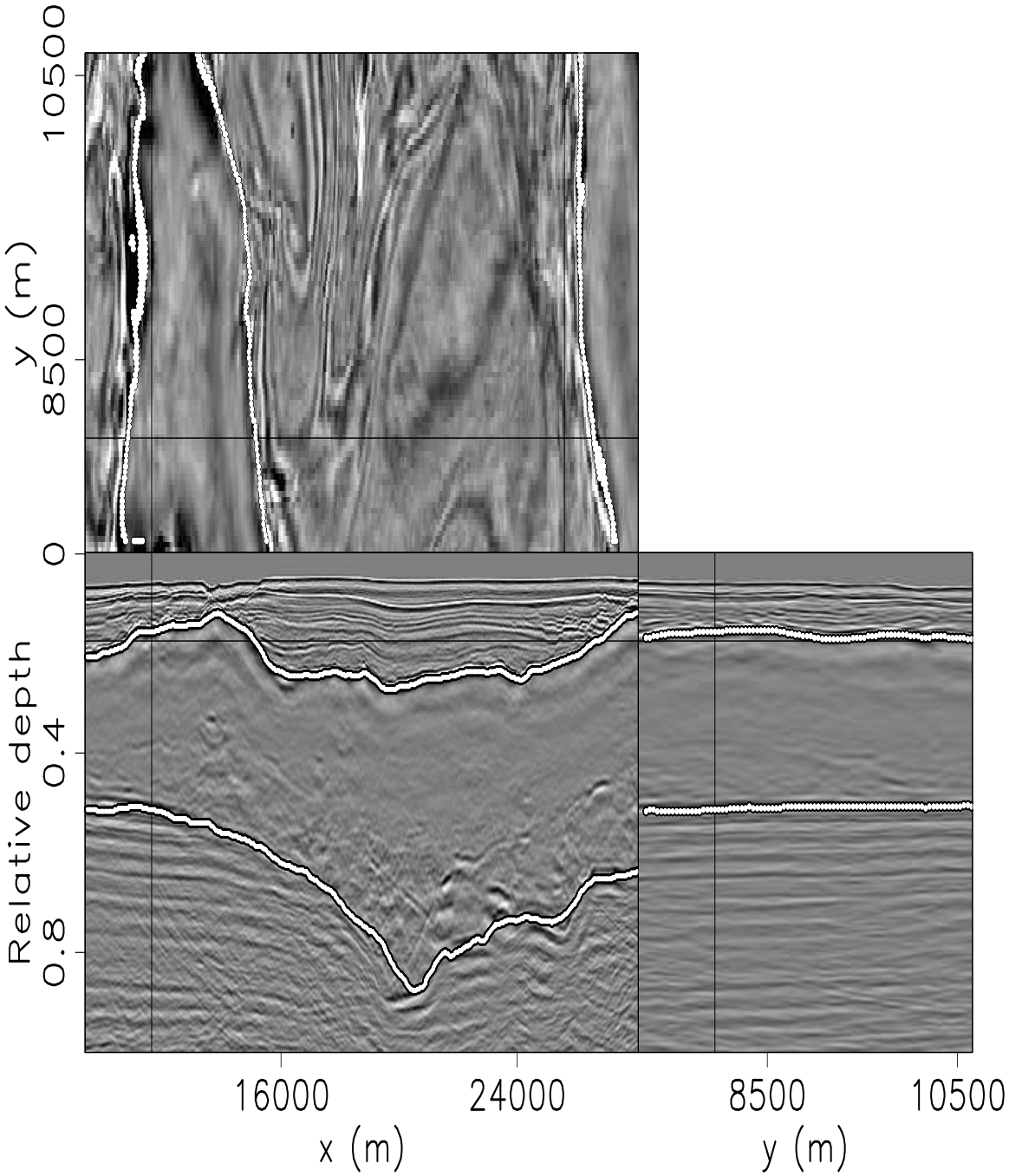

Figure 3.19: The same image as in Figure3.18, except a depth slice at relative depth=0.176 , an in-line section at y=7960 m, and a cross-line section at x=11625 m. segment-amp_pck1b_bw [CR]

data with the resulting salt-interface pick found by clipping the eigenvector at the zero contour. Notice it does a good job of tracking the peak of the amplitude, even on the lower interface. In practice, it is not important to pick the lower interface until the upper interface has been well imaged. Recall that this is a 3D picking solution using the entire 3D dataset within bounds as seen in Figure 3.16. In the depth slice on the top of the figure, it is clear that the tracking is accurate. Different slices through the same volume are shown in Figure 3.19. This favorable result was found simply using instantaneous amplitude with very little parameter adjustment.

In Figure 3.20 is a windowed portion of Figure 3.18 on the upper salt interface, where a steep canyon is mispicked by the segmentation method using instantaneous amplitude alone. A manually picked salt interface is displayed for comparison in Figure 3.21. The manual pick is, in general below the segmentation pick because the interpreter intentionally picks below the interface so that the migration will place the wavelet in sediment velocities to minimize stretching. On the lower interface, an interpreter may pick slightly above the interface. Nevertheless, the two picks are in agreement in most places but noticably different in the canyon.

Inspection of the instantaneous amplitude in Figure 3.22 reveals the source of the error. The picking result is tracking a bright amplitude reflector as it should, but unfortunately this bright reflector is not the salt interface.

Additional processing of the instantaneous amplitude in Figure 3.22 only slightly improves the result. I applied spherical AGC (see Figure 3.23) to balance the amplitudes along the interface. Spherical (or 3D) AGC is like standard AGC, but instead of scanning a one dimensional time window, it scans a three dimensional sphere. In this example, the sphere has a radius of 25 samples. To avoid amplifying events other than the salt interface, the size of the radius of the sphere used for AGC should always be greater than the radius of the bounded image. The resulting second smallest eigenvector using the instantaneous amplitude with spherical AGC is shown in Figure 3.24. I selected a contour value of -.00003 to attempt to force the result to pick correctly in the canyon (see Figure 3.25). Unfortunately, by selecting the appropriate contour in the vicinity of the canyon, I create errors in other areas. In areas where the eigenvector is not smooth, it is less sensitive to changes in the splitting point.

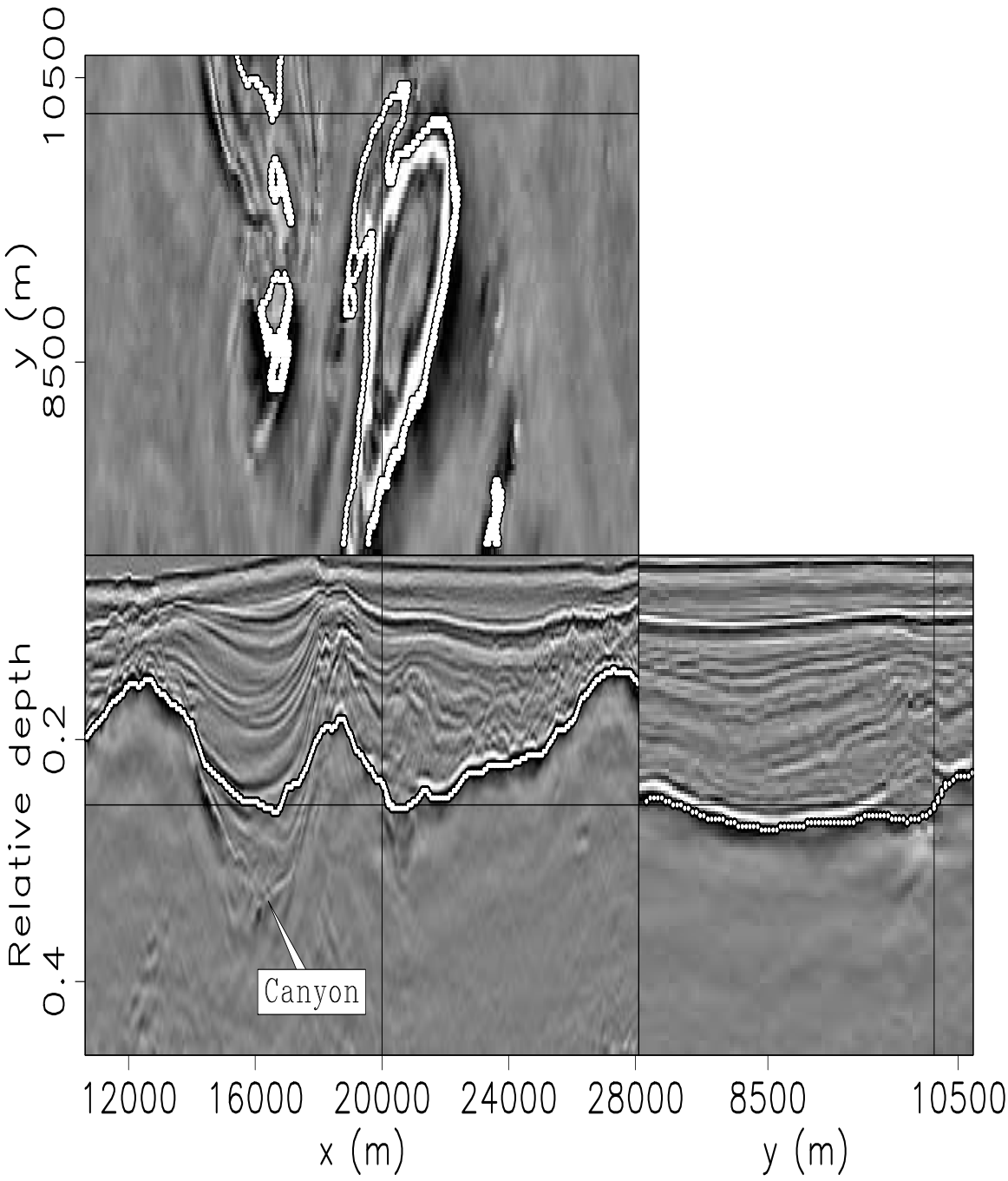

Figure 3.20: A windowed portion on the upper salt interface of Figure 3.18 where the method inaccurately picks a canyon. The black lines superimposed onto the orthogonal sections identify the location of these sections: a depth slice at relative depth=0.26 , an in-line section at y=10320 m, and a cross-line section at x=20025 m. segment-amp_pck1_win_bw [CR]

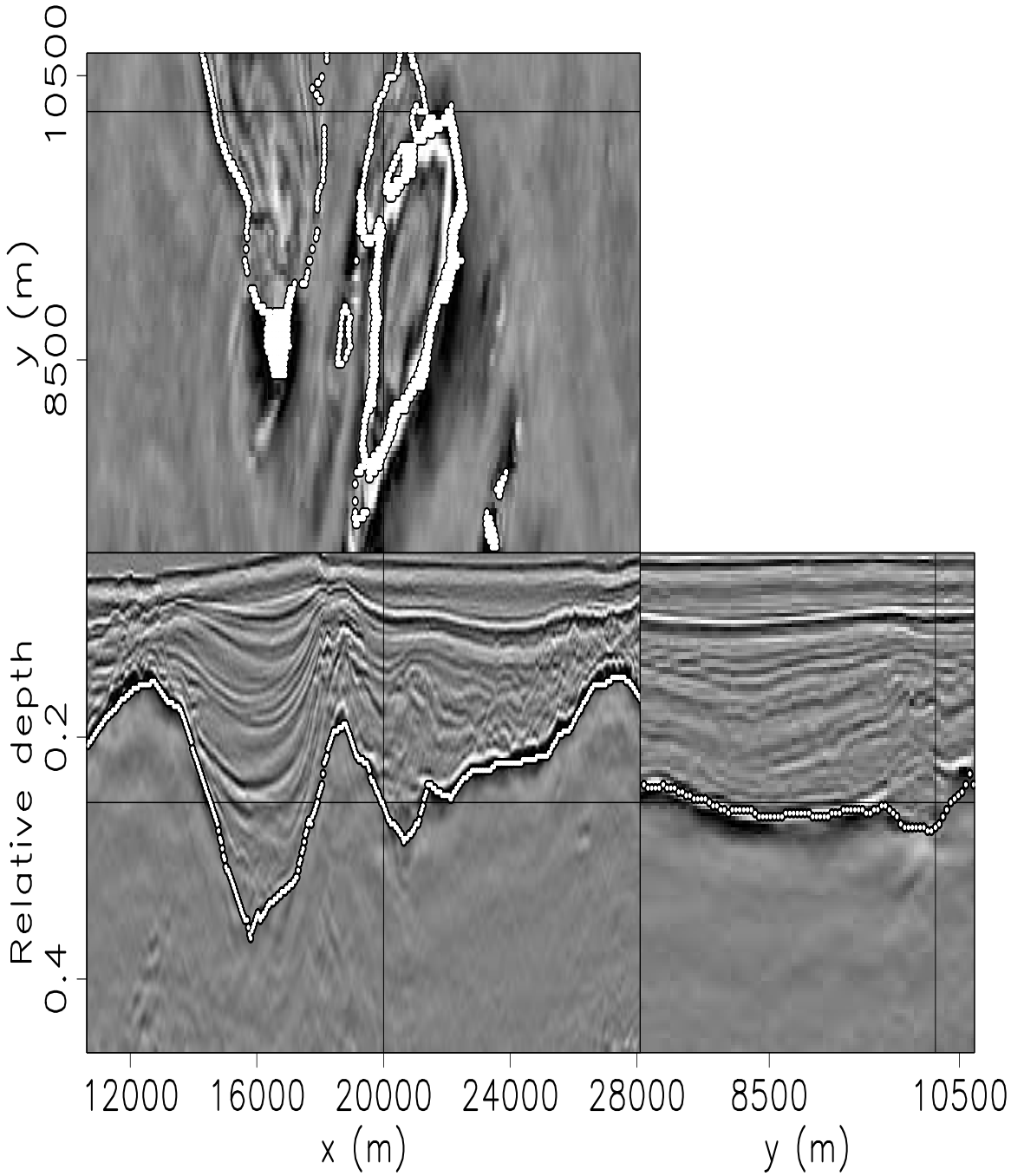

Figure 3.21: The same image as in Figure 3.20, but showing a manually picked interface for comparison. segment-man_pck1_win_bw [CR]

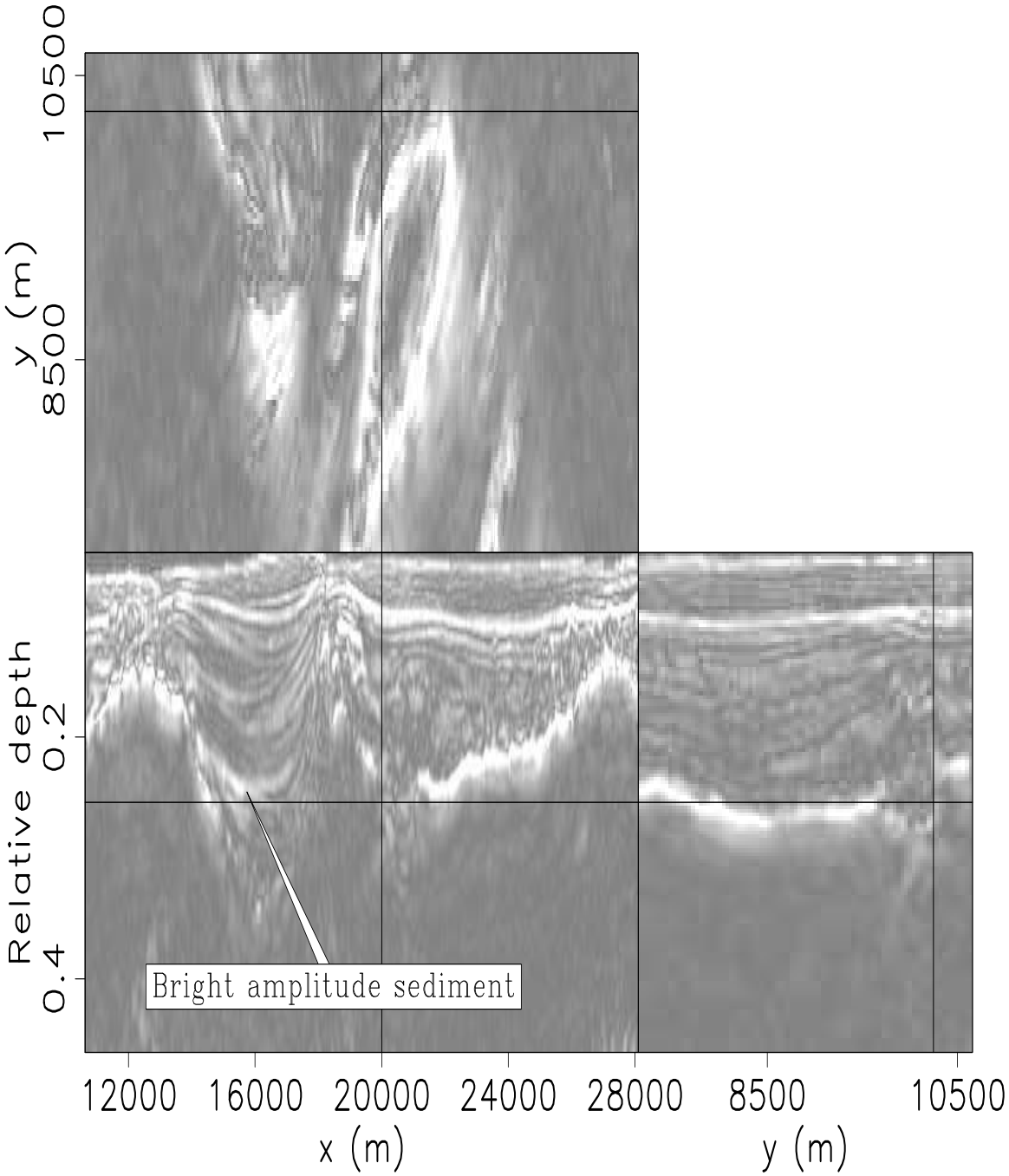

Figure 3.22: The same image as in Figure 3.20, but showing instantaneous amplitude. It is clear that instantaneous amplitude does not well define the salt interface in the canyon. segment-amp_win_new [CR]

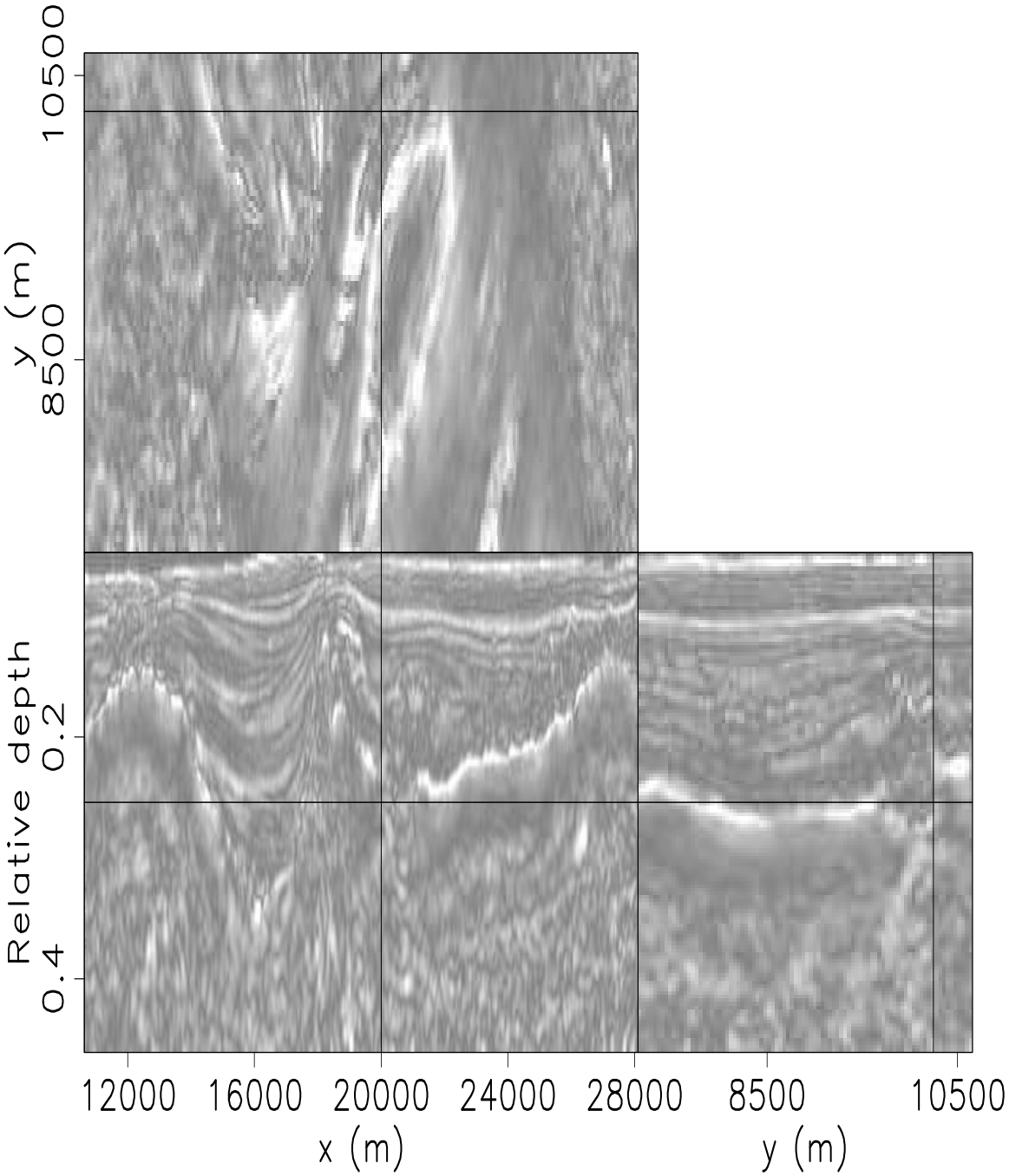

Figure 3.23: The same image as in Figure 3.20, but showing instantaneous amplitude with AGC. This does a better job of defining the salt interface in the canyon but increases the amplitude of other reflectors. segment-amp_AGC_win_new [CR]

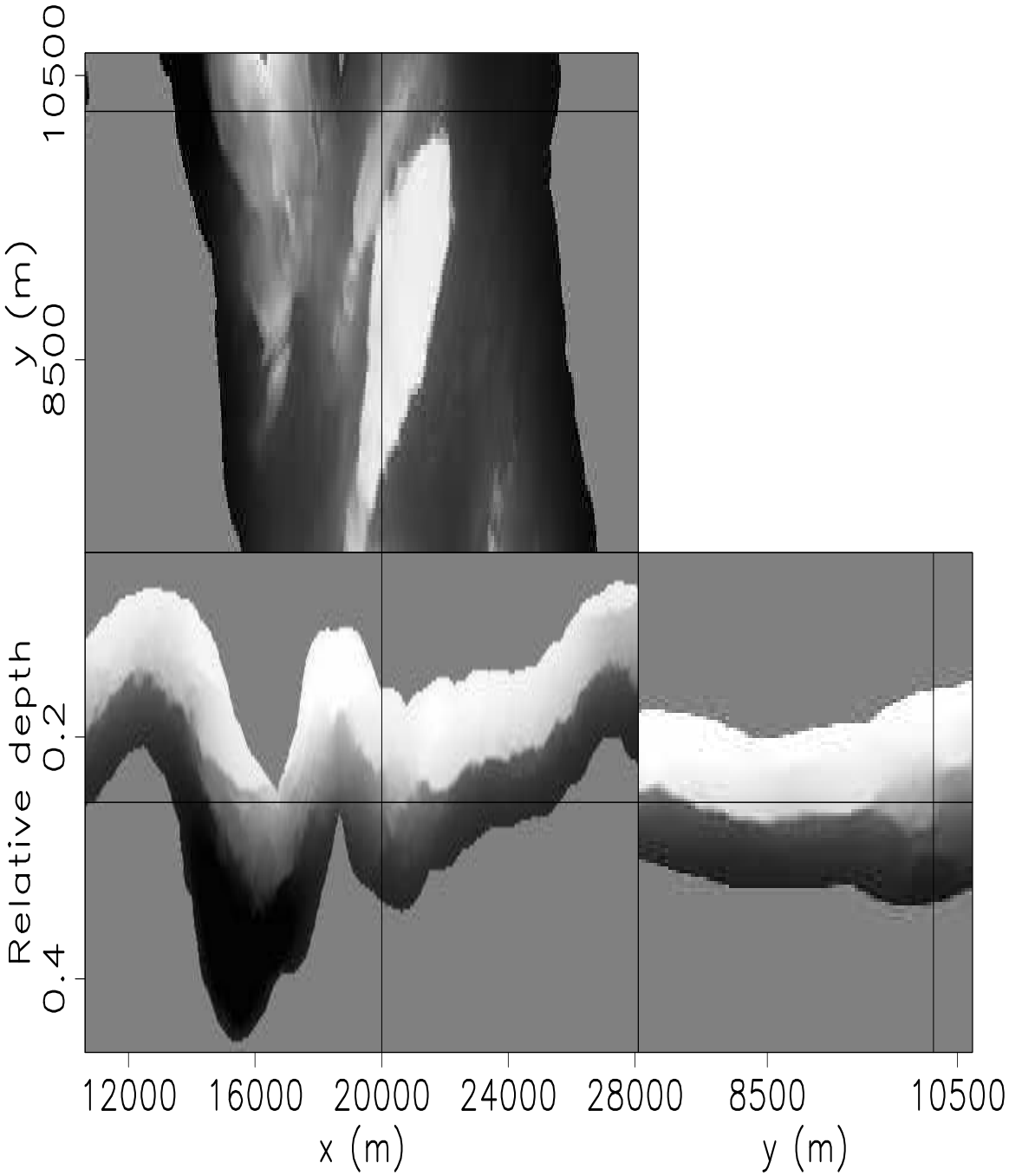

Figure 3.24: The same image as in Figure 3.20, only showing the eigenvector created using the instantaneous amplitude with spherical AGC shown in Figure 3.23. segment-amp_AGC_eig1_win_new [CR]

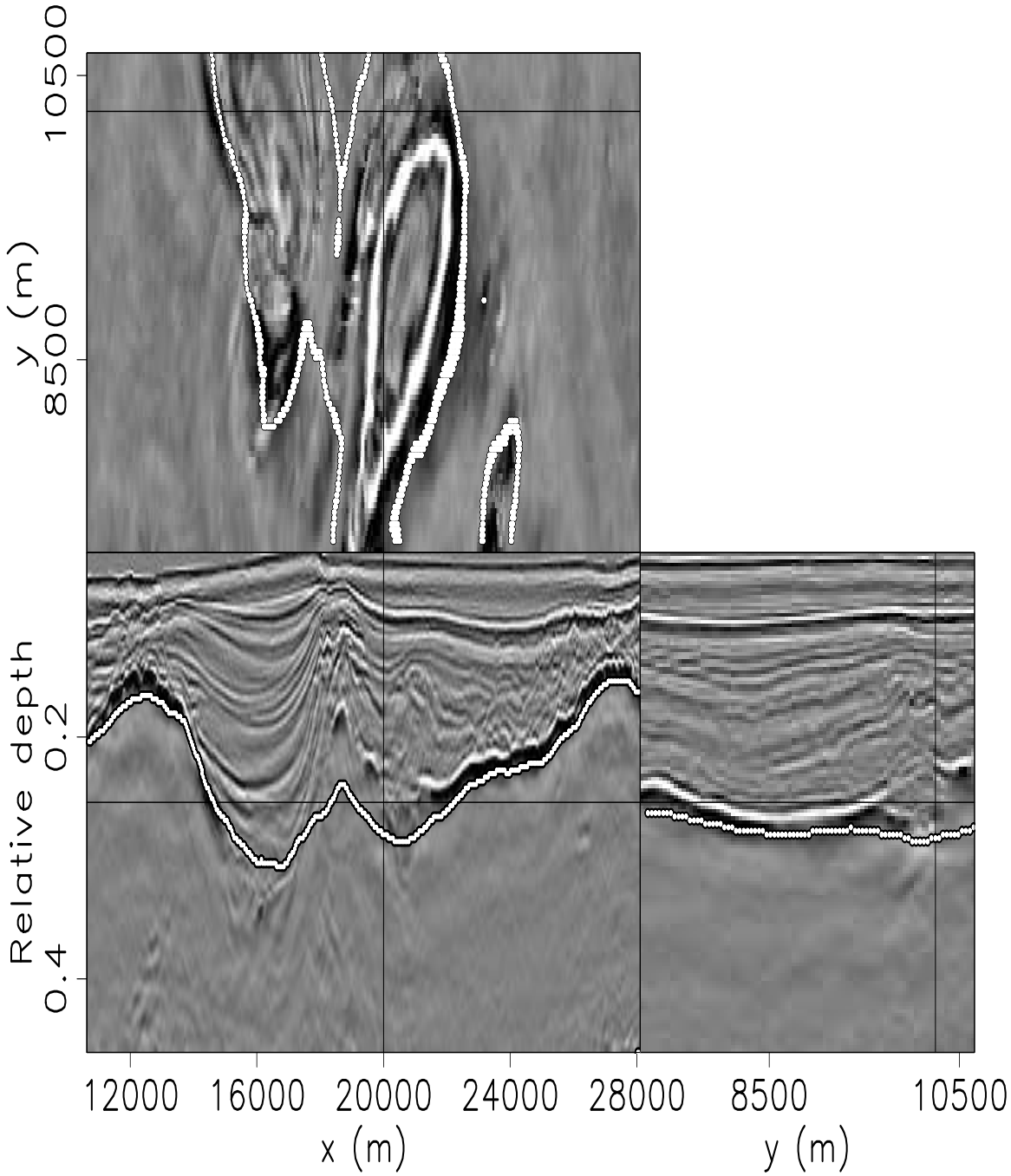

Figure 3.25: The same image as in Figure 3.20, but splitting the eigenvector on a non-zero contour value (-.00003) of the eigenvector. Any improvement in the canyon is matched by a reduction in picking quality in other uncertain areas. segment-amp_pck2_win_bw [CR]

Dip variability attribute

To improve the picking result I investigate using dip variation and amplitude together. The dip variation attribute is a good salt indicator, because a salt interface often has different dip than adjacent reflectors, notwithstanding that the dip within a salt body is often different than the dip of adjacent reflectors. To generate a dip-variability attribute volume, I first estimate dip (Fomel, 2002) in the x and y directions. I then apply the helical derivative (Claerbout, 1999) to highlight changes in dip in three dimensions. I then take the envelope of both dip-derivative volumes and combine them. Lastly, I multiply this dip variability attribute by the instantaneous amplitude with spherical AGC to get the attribute displayed in Figure 3.26. I then use this attribute to estimate the eigenvector displayed in Figure 3.27. This result was calculated using the weight criterion in equation 3.8 with $\epsilon_A = .1$ and $\epsilon_X = 8.0$. Again, I ignore the distance weight by using a large ϵ_X. I used the zero contour to pick the interface displayed in Figure 3.28. It is clear that this picking result is very close to the manually pick in Figure 3.21. Clearly there are some minor differences that could be manually corrected. Inspection of the depth slice reveals the picking is consistent in 3D. Other slices through the same volume are shown in Figure 3.29. Furthermore, a perspective view of the tracking result of this method reveals a consistent picking result (see Figure 3.30).

DISCUSSION

Improved picking results are simply a matter of identifying the attribute (or set of attributes) that most accurately locates the salt interface. I recommend first windowing a small piece of the salt interface, preferably containing a range of picking challenges, then testing several attributes to find the best combination.

Shi and Malik (2000) recommend finding the optimum splitting point by searching the eigenvector for the best N_{cut}. Thus far my experience has shown that by manually splitting the eigenvector to improve the solution in one place can cause errors in other places. However, it does seem plausible to have a smoothly changing splitting point from place to place. This splitting point could be chosen by an interpreter.

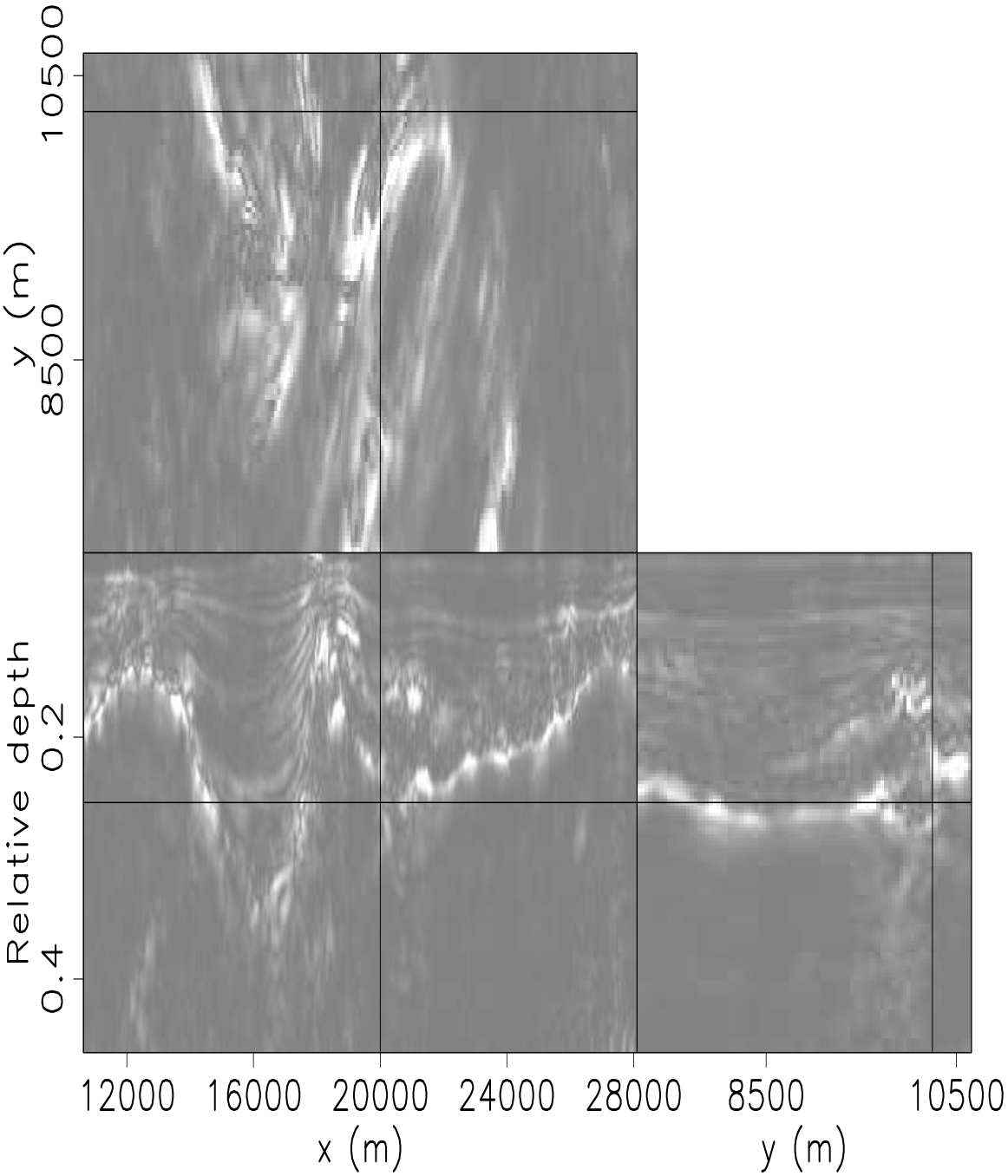

Figure 3.26: The same image as in Figure 3.22, but showing the combined attributes from dip and instantaneous amplitude. segment-combo_attr_new [CR]

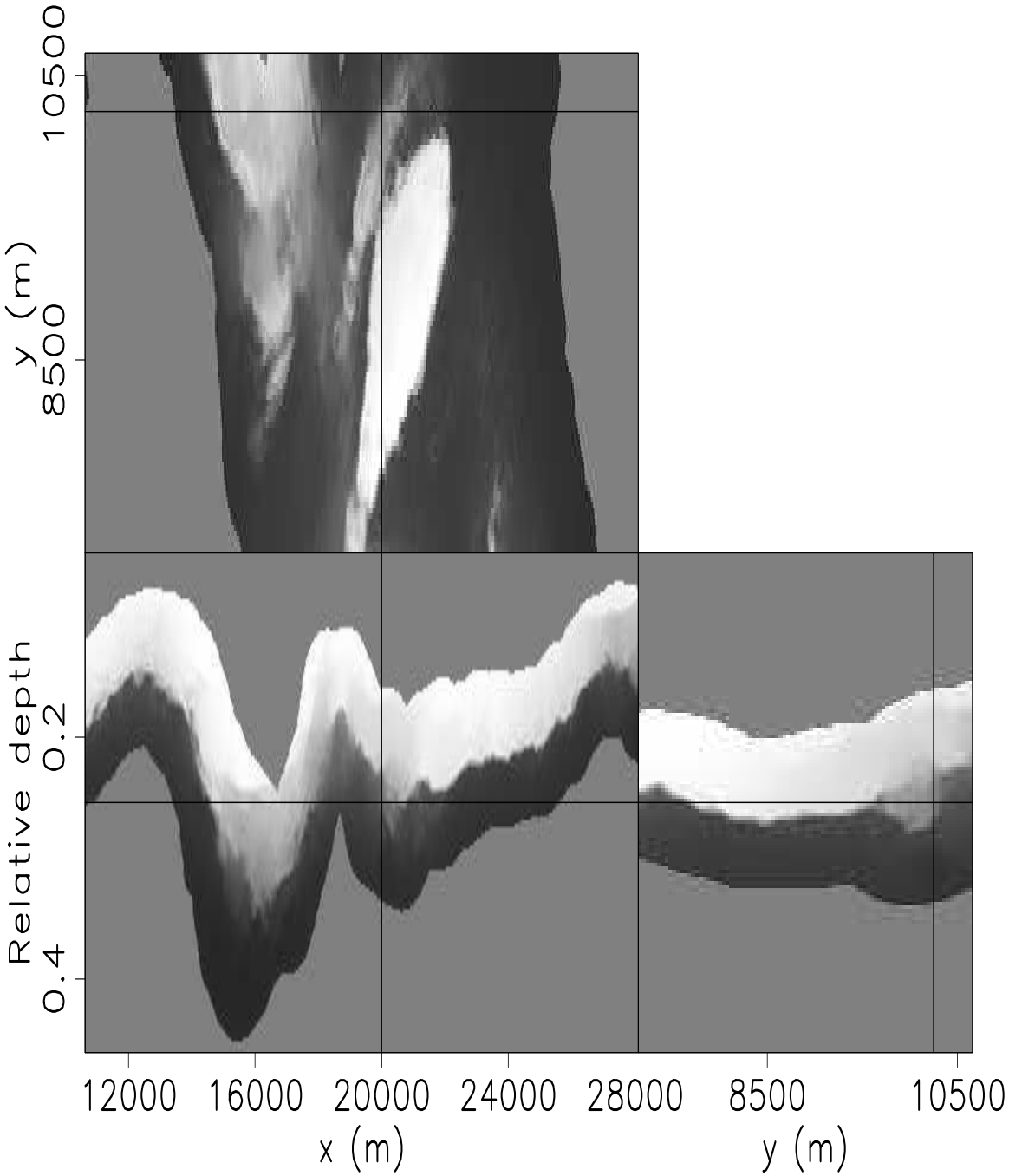

Figure 3.27: The eigenvector resulting from the combined attributes in Figure 3.26. segment-combo_eig1_win_new [CR]

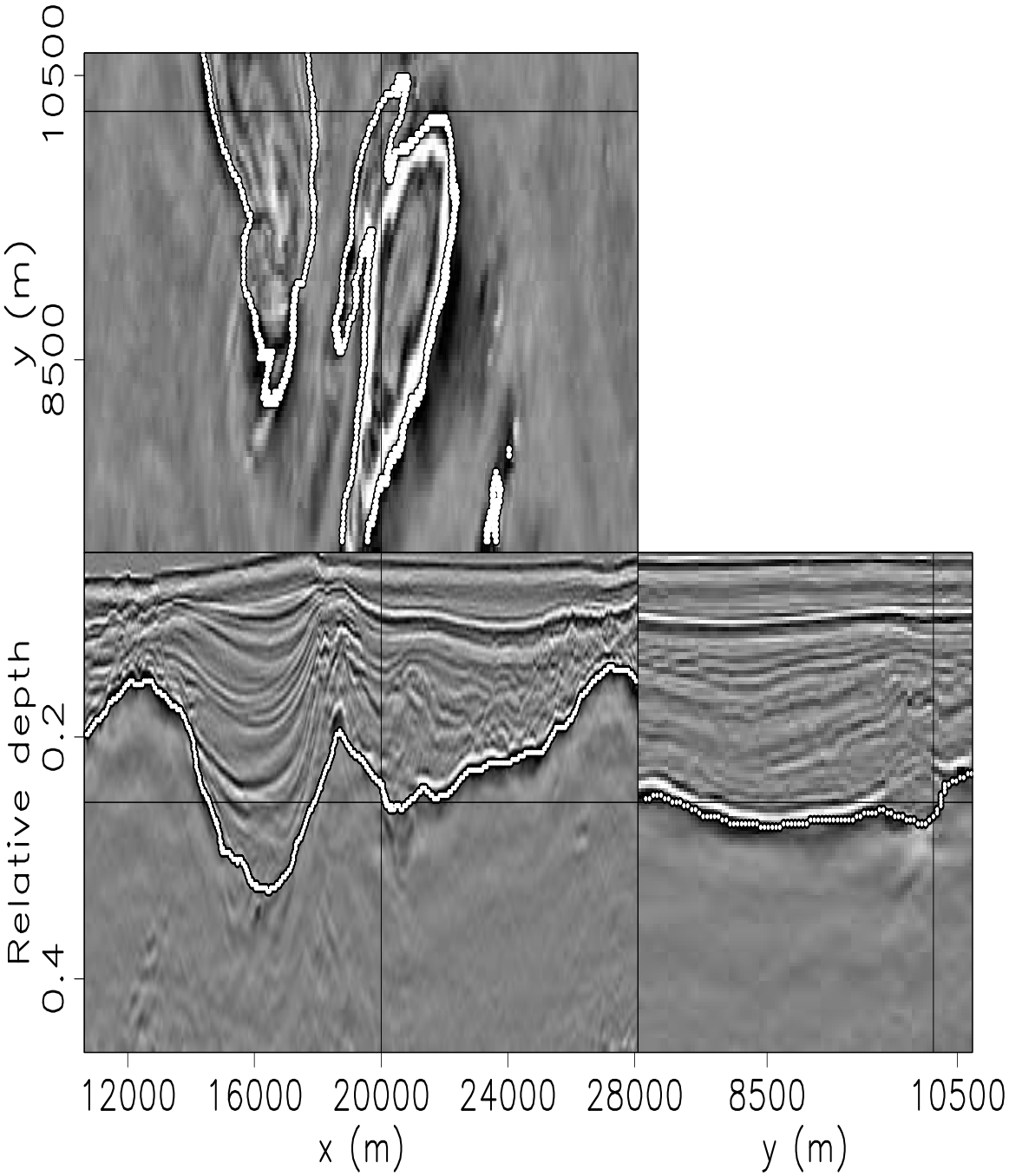

Figure 3.28: The interface is extracted from the image in Figure 3.27 and overlain on the instantaneous amplitude of data. It accurately tracks the interface. A manually picked interface is shown in Figure 3.21 for comparison. segment-combo_pck1_win_bw [CR]

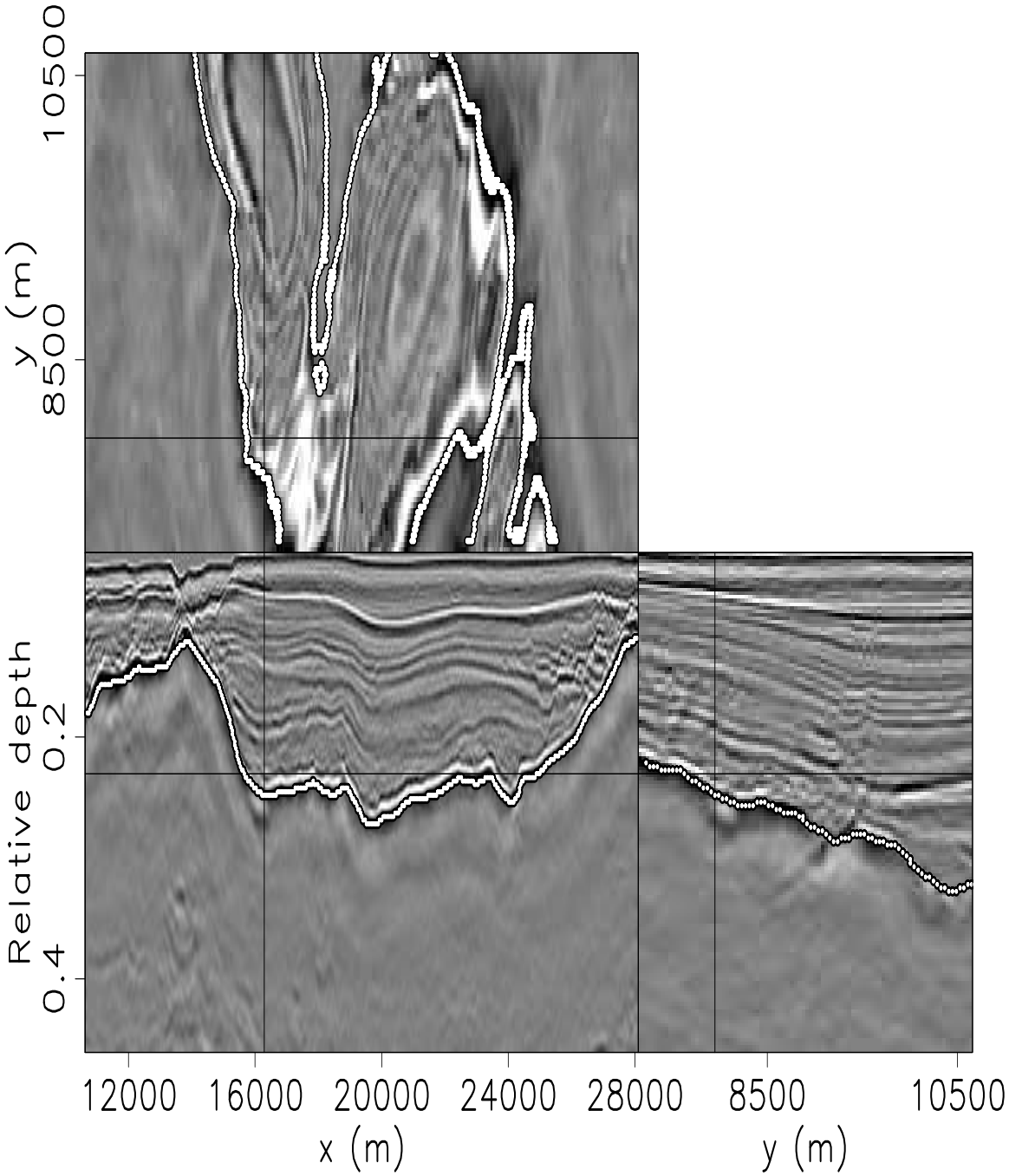

Figure 3.29: The same image as in Figure 3.28, except a depth slice at relative depth=0.23 , an in-line section at y=7960 m, and a cross-line section at x=16312.5 m. segment-combo_pck1_winb_bw [CR]

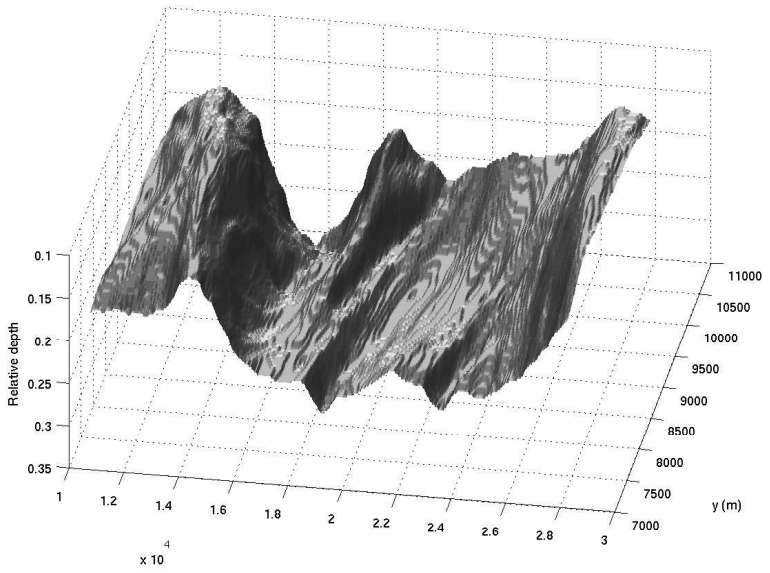

Figure 3.30: A perspective view of the picking result shown in Figures 3.28 and 3.29. segment-image_surf [CR]

Segmentation has the potential ability to track reflections other than salt boundaries. The salt-boundary application is attractive because the salt interface typically has the brightest amplitude in the images, which makes it easy to make the weights (used to segment the image) low at the salt interface. However, in principle, attributes can be generated to make weights the weakest at other reflections. This can be achieved by cross-correlation, selecting a seed point on a trace at a reflector of interest, and correlating all of the surrounding traces with that trace. As long as the wavelet character doesn't change too much, this resulting correlation field can be used as input into the segmentation algorithm to partition the image along the boundary identified by the seed point. This has the potential to robustly pick across faults.

CONCLUSIONS

In this chapter, I have presented a modification of NCIS for tracking 3D seismic salt boundaries. This algorithm provides a globally optimized picking solution that is less prone to errors

caused by local discontinuities and amplitude fluctuations than traditional autotrackers. Also, since amplitude alone may not fully define the salt interface, this algorithm has the flexibility to incorporate any number of attributes, while fully exploiting the dimensionality of the data. In some cases, this allows the quality of the picking results to approach (or exceed) that of human manual interpretation.

Several cost-saving measures have been presented to make this a practical and affordable tool. Narrow bounds can be imposed to reduce the effective problem size and increase robustness. I also parallelized two key steps of the algorithm, the calculation of the weight matrix and the eigenvector, to take advantage of modern parallel clusters.

Chapter 4

Conclusions

In this thesis, I present both flattening and image segmentation algorithms applied to the problem of 3D seismic interpretation. Both algorithms exploit the full dimensionality of 3D seismic data to generate global interpretation solutions. Both algorithms have been implemented in a computationally practical way. Furthermore, both algorithms have the capability of incorporating manual interpretation. In the case of flattening, manual picks can be imposed as hard constraints. In the case of segmentation, manual bound-constraints can be imposed. Even more, the eigenvector used in segmentation contains information about the certainty of the picking solution and can be used to direct a human interpreter to challenging locations.

Intuitively, it seems that there are many ways to combine these two methods together. For instance, flattening could be used to generate an attribute that could be used to segment an image. Alternatively, segmentation could be used to segment cubes into regions that are flattened separately, similar to the implementation of Hale and Emanuel (2003) for painting atomic meshes into reservoir tanks.

Both methods have significant applicability in the world of seismic data analysis that have not been fully realized. The ability to add constraints to flattening opens the door on directions of further development. One of the most obvious is iterative flattening. This means that flattening can be run repeatedly. After each run, the algorithm can be designed to select portions of the resulting $\boldsymbol{\tau}$ field to be used as constraints in the subsequent runs of flattening. A suitable

overall measure of flatness is total stacked energy. The constraints also will make flattening a tool that could easily be incorporated into an interpretation work-flow. I envision that with additional computational improvements, multiple flattening passes can be run quickly as an interpreter adds picks (constraints). On the other hand, the segmentation method has unrealized uses as well. For instance, it has the potential to pick other reflections besides salt boundaries.

Both methods utilize greater dimensionality than humans can handle alone and also provide a framework in which the human can play a major role. They are global methods and are contributing to the technological movement toward bringing more to the computer's strength in dealing with higher dimensionality to bear on the historically manual process of extraction of geological information from seismic data volumes.

Chapter 5

Euler-Lagrange solution for flattening

The Euler-Lagrange equation (Farlow, 1993) can be applied to the flattening problem (developed by Sergey Fomel) to find the function $\boldsymbol{\tau}(x,y,t)$ that minimizes

$$J(\boldsymbol{\tau}) = \int\int F(x,y,\boldsymbol{\tau},\boldsymbol{\tau}_x,\boldsymbol{\tau}_y)\,dx\,dy \approx 0, \tag{5.1}$$

where $\boldsymbol{\tau}_x$ and $\boldsymbol{\tau}_y$ are the derivatives of $\boldsymbol{\tau}$ in the x and y directions, respectively, and is given by

$$\frac{\partial F}{\partial \boldsymbol{\tau}} - \frac{d}{dx}\left[\frac{\partial F}{\partial \boldsymbol{\tau}_x}\right] - \frac{d}{dy}\left[\frac{\partial F}{\partial \boldsymbol{\tau}_y}\right] = 0. \tag{5.2}$$

In this application, I have

$$J(\boldsymbol{\tau}) = \int\int \left[\left(\mathbf{b}(x,y,\boldsymbol{\tau}) - \frac{\partial \boldsymbol{\tau}}{\partial x}\right)^2 + \left(\mathbf{q}(x,y,\boldsymbol{\tau}) - \frac{\partial \boldsymbol{\tau}}{\partial y}\right)^2\right] dx\,dy, \tag{5.3}$$

where $\mathbf{b}$ is the dip in the x direction and $\mathbf{q}$ is the dip in the y direction. Calculating $\frac{\partial F}{\partial \boldsymbol{\tau}}$, $\frac{\partial F}{\partial \boldsymbol{\tau}_x}$, and $\frac{\partial F}{\partial \boldsymbol{\tau}_y}$, I find

$$\frac{\partial F}{\partial \boldsymbol{\tau}} = 2\left[\mathbf{b} - \frac{\partial \boldsymbol{\tau}}{\partial x}\right]\frac{\partial \mathbf{b}}{\partial \boldsymbol{\tau}} + 2\left[\mathbf{q} - \frac{\partial \boldsymbol{\tau}}{\partial y}\right]\frac{\partial \mathbf{q}}{\partial \boldsymbol{\tau}}, \tag{5.4}$$

$$\frac{\partial F}{\partial \boldsymbol{\tau}_x} = -2\left[\mathbf{b} - \frac{\partial \boldsymbol{\tau}}{\partial x}\right] \tag{5.5}$$

and

$$\frac{\partial F}{\partial \boldsymbol{\tau}_y} = -2\left[\mathbf{q} - \frac{\partial \boldsymbol{\tau}}{\partial y}\right]. \tag{5.6}$$

Then substituting equations (5.4), (5.5), and (5.6) into equation (5.2) and simplifying, I get

$$\frac{\partial^2 \boldsymbol{\tau}}{\partial x^2} + \frac{\partial^2 \boldsymbol{\tau}}{\partial y^2} = \frac{\partial \mathbf{b}}{\partial x} + \frac{\partial \mathbf{q}}{\partial y} + \frac{1}{2}\frac{\partial(\mathbf{b}^2 + \mathbf{q}^2)}{\partial \boldsymbol{\tau}}. \tag{5.7}$$

In the method, I ignore the last term of equation (5.7) and iteratively solve

$$\frac{\partial^2 \boldsymbol{\tau}}{\partial x^2} + \frac{\partial^2 \boldsymbol{\tau}}{\partial y^2} = \frac{\partial \mathbf{b}}{\partial x} + \frac{\partial \mathbf{q}}{\partial y}. \tag{5.8}$$

I choose to ignore the last term because the first two terms define a gradient direction which I can implement in the Fourier domain efficiently. Without the third term, the method takes more iterations to reach the same answer, because the gradient calculation is less accurate. However, because each iteration can be performed much more quickly if the third term is not included, it is more computationally efficient to neglect it. Equation (5.8) can be rewritten using the gradient ($\boldsymbol{\nabla} = [\frac{\partial}{\partial x} \;\; \frac{\partial}{\partial y}]^T$) and the estimated dip ($\mathbf{p} = [\mathbf{b} \;\; \mathbf{q}]^T$) as

$$\boldsymbol{\nabla}^T \boldsymbol{\nabla} \boldsymbol{\tau} \quad = \quad \boldsymbol{\nabla}^T \mathbf{p}. \tag{5.9}$$

Therefore, the regression to be solved is

$$\boldsymbol{\nabla} \boldsymbol{\tau} \quad = \quad \mathbf{p}. \tag{5.10}$$

Chapter 6

Preconditioning with the helical derivative

Instead of using the cosine transform method as a preconditioner, I can precondition with the helical derivative (Claerbout, 1999). This is done by preconditioning by substituting $\boldsymbol{\tau} = \mathbf{H}^{-1}\mathbf{m}$, where $\mathbf{H}^{-1}$ is the inverse of the 3D helical derivative. However, recall that the 3D gradient operator $\boldsymbol{\nabla}_\epsilon$ is actually a chain of two matrices $\mathbf{W}_\epsilon$ and $\boldsymbol{\nabla}$. Instead of approximating the inverse of $(\boldsymbol{\nabla}^T\boldsymbol{\nabla})^{-1}$, I wish to approximate the inverse of $(\boldsymbol{\nabla}^T{\mathbf{W}_\epsilon}^2\boldsymbol{\nabla})^{-1}$. Therefore, I factor the finite difference approximation to the Laplacian with an ϵ parameter. In 2D ((x,t)-plane), I factor the Laplacian with an epsilon parameter on the t-axis as:

	$-\epsilon^2$	
-1	$2+2\epsilon^2$	-1
	$-\epsilon^2$	

(6.1)

To extend to 3D, I add another 2 to the center and another set of -1's in the 3rd dimension. Once factored it becomes a 3D helical derivative $\mathbf{H}_\epsilon$ with a scalar weight, ϵ, applied to the time(or depth) axis. When used as a regularization operator, this has the desirable property of having only one output. I choose not to include the mask $\mathbf{K}$ in the preconditioner because it is non-stationary.

With preconditioning, equation (2.36) becomes this:

$$\mathbf{r} \quad = \quad \nabla_\epsilon \mathbf{K} \mathbf{H}_\epsilon{}^{-1} \mathbf{m} - \mathbf{p}_\epsilon(x, y, \boldsymbol{\tau}_k) + \mathbf{r}_0. \tag{6.2}$$

A significant price is paid in memory cost for this computational time saving. To precondition this equation requires chaining together several operators thus several temporary arrays are allocated. Also, the computational expense is tied to the number of coefficients used in the filter which in 3D is typically 20.

Chapter 7

Quasi-Newton constrained flattening

Alternatively, I can use the L-BFGS-B algorithm (Guitton et al., 2005b) for imposing very tight constraints on the picked values of the tau field. The L-BFGS-B algorithm seeks to find a vector of model parameters $\boldsymbol{\tau}$ such that I minimize

$$\min f(\boldsymbol{\tau}) \text{ subject to } \boldsymbol{\tau} \in \Omega, \tag{7.1}$$

where

$$\boldsymbol{\tau} \in \Omega = \{\boldsymbol{\tau} \in \Re^N \mid l_i \leq \boldsymbol{\tau}_i \leq u_i\}, \tag{7.2}$$

with l_i and u_i being the lower and upper bounds for the model $\boldsymbol{\tau}_i$, respectively. In this case, l_i and u_i are called simple bounds. For the flattening technique, I want to minimize (Lomask, 2003b; Guitton et al., 2005a; Lomask et al., 2006b)

$$f(\boldsymbol{\tau}) = \int\int \left[\left(b(x,y,z;\boldsymbol{\tau}) - \frac{\partial \boldsymbol{\tau}}{\partial x} \right)^2 + \left(q(x,y,z;\boldsymbol{\tau}) - \frac{\partial \boldsymbol{\tau}}{\partial y} \right)^2 \right] dx\, dy, \tag{7.3}$$

Here, $\mathbf{b}$ is the dip in the x direction and $\mathbf{q}$ is the dip in the y direction. The L-BFGS-B algorithm combines a quasi-Newton update of the Hessian (second derivative) with a trust-region method.

Incorporating the initial $\boldsymbol{\tau}$ field is trivial with the L-BFGS-B method: I simply set the

bounds where an a-priori value exists:

$$l_i = \boldsymbol{\tau}_{0_i} - \alpha \times \boldsymbol{\tau}_{0_i} \tag{7.4}$$

$$u_i = \boldsymbol{\tau}_{0_i} + \alpha \times \boldsymbol{\tau}_{0_i}, \tag{7.5}$$

where α is a small number (≈ 0.001). Note that the L-BFGS-B algorithm allows us to optionally activate the constraints for every point of the model space. Note that in equation 7.3, the objective function incorporates a smoothing in the vertical direction of the $\boldsymbol{\tau}$ field as well.

Bibliography

Bahorich, M. and S. Farmer, 1995, The coherence cube: The Leading Edge,1053–1058.

Bienati, N. and U. Spagnolini, 1998, Traveltime picking in 3D data volumes: 60th European Association of Geoscientists & Engineers Meeting, Extended Abstracts, Session:01-12.

Bienati, N. and U. Spagnolini, 2001, Multidimensional wavefront estimation from differential delays: Institute of Electrical and Electronics Engineers Transactions on Geoscience and Remote Sensing, **39**, no. 3, 655–664.

Bienati, N., M. Nicoli, and U. Spagnolini, 1999a, Automatic horizon picking algorithms for multidimensional data: 61st European Association of Geoscientists & Engineers Meeting, Helsinki, Extended Abstracts, Session:6030.

Bienati, N., M. Nicoli, and U. Spagnolini, 1999b, Horizon picking for multidimensional data: an integrated approach: Proceedings: 6th International Congress of the Brazilian Geophysical Society, Rio de Janeiro, SBGf359.

Blinov, A. and M. Petrou, 2003, Reconstruction of 3D horizons from 3D seismic data sets: International Society for Optical Engineering 10th International Symposium on Remote sensing, Remote Sensing for environmental monitoring, GIS applications and Geology III.

Brown, M. P., S. A. Morton, and G. Whittle, 2006, Seismic event tracking by global path optimization: 76th Annual International Meeting, SEG, Expanded Abstracts.

Claerbout, J. F. and F. Muir, 1973, Robust modeling with erratic data: Geophysics, **38**, no. 5, 826–844.

Claerbout, J. F., 1992, Earth Soundings Analysis: Processing Versus Inversion: Blackwell Scientific Publications.

Claerbout, J., 1999, Geophysical estimation by example: Environmental soundings image enhancement: Stanford Exploration Project, `http://sepwww.stanford.edu/sep/prof/`.

Clapp, R. G., 2005, Inversion and fault tolerant parallelization using python: Stanford exploration project report, **120**, 41–62.

Farlow, S. J., 1993, Partial differential equations for scientists and engineers: Dover Publications.

Fomel, S., 2002, Applications of plane-wave destruction filters: Geophysics, **67**, no. 6, 1946–1960.

Gdalyahu, Y., D. Weinshall, and M. Werman, 2001, Self organization in vision: stochastic clustering for image segmentation, perceptual grouping, and image database organization: Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence, **23**, no. 10, 1053–1074.

Ghiglia, D. C. and M. D. Pritt, 1998, Two-dimentional phase unwrapping: Theory, algorithms, and software: John Wiley and Sons, Incorporated.

Ghiglia, D. C. and L. A. Romero, 1994, Robust two-dimensional weighted and unweighted phase unwrapping that uses fast transforms and iterative methods: Optical Society of America, **11**, no. 1, 107–117.

Golub, G. H. and C. F. V. Loan, 1989, Matrix computations: John Hopkins Press.

Guitton, A., J. F. Claerbout, and J. M. Lomask, 2004, First order interval velocity estimates without picking: 74th Society of Exploration Geophysicists Meeting, Extended Abstracts, 2339–2342.

Guitton, A., J. M. Lomask, and S. Fomel, 2005a, Non-linear estimation of vertical delayx: 75th SEG Meeting, Extended Abstracts.

Guitton, A., J. Lomask, and S. Fomel, 2005b, Non-linear estimation of vertical delays with a quasi-Newton method: SEP–**120**, 167–178.

Hale, D. and J. U. Emanuel, 2002, Atomic meshing of seismic images: SEG, Expanded Abstracts, 2126–2129.

Hale, D. and J. U. Emanuel, 2003, Seismic interpretation using global image segmentation: SEG, Expanded Abstracts, 2410–2413.

Hanning, T. and G. M. Pisinger, 2002, A pixel-based segmentation algorithm of color images by n-level-fitting: 5th international association of science and technology for development, Computer graphics and imaging.

James, H., A. Peloso, and J. Wang, 2002, Volume interpretation of multi-attribute 3D surveys: First Break, **20**, no. 3, 176–180.

Lee, R., 2001, Pitfalls in seismic data flattening: The Leading Edge, **20**, no. 01, 160–164.

Leggett, M., W. A. Sandham, and T. S. Durrani, 1996, 3D horizon tracking using artificial neural networks: First Break, **14**, no. 11, 413–418.

Lehoucq, R. B. and J. A. Scott, 1996, An evaluation of software for computing eigenvalues of sparse nonsymmetric matricies: Preprint MCS-P547-1195, Argonne National Laboratory, Argone III.

Lomask, J. and B. Biondi, 2003, Image segmentation for tracking salt boundaries: Stanford exploration project report, **114**, 193–200.

Lomask, J. and J. Claerbout, 2002, Flattening without picking: SEP–**112**, 141–150.

Lomask, J. and A. Guitton, 2006a, Flattening with geological constraints: SEP–**124**.

Lomask, J. and A. Guitton, 2006b, Flattening with geological constraints: 76th SEG Meeting, Extended Abstracts, 1053–1056.

Lomask, J., B. Biondi, and J. Shragge, 2004, Image segmentation for tracking salt boundaries: 74th Annual International Meeting, SEG, Expanded Abstracts.

Lomask, J., A. Guitton, S. Fomel, and J. Claerbout, 2005, Update on flattening without picking: SEP–**120**, 137–158.

Lomask, J., R. G. Clapp, and B. Biondi, 2006a, Parallel implementation of image segmentation for tracking 3d salt boundaries: 68th Annual International Meeting, EAGE, Expanded Abstracts.

Lomask, J., A. Guitton, S. Fomel, and J. Claerbout, 2006b, Flattening without picking: Geophysics, **71**, no. 4, 13–20.

Lomask, J., R. G. Clapp, and B. Biondi, submitted 2006, Image segmentation for tracking 3d salt boundaries: Geophysics.

Lomask, J., 2002, Fault contours from seismic: SEP–**111**, 295–309.

Lomask, J., 2003a, Flattening 3D seismic cubes without picking: 73th Society of Exploration Geophysicists Meeting, Extended Abstracts, 1402–1405.

Lomask, J., 2003b, Flattening 3-D data cubes in complex geology: SEP–**113**, 247–260.

MacQueen, J. B., 1967, Some methods for classification and analysis of multivariate observations: Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, 281–297.

Maerten, L. and F. Maerten. Poly3d and dynel:. Internet, 2002. http://pangea.stanford.edu/research/geomech/Software/Software.htm.

Maerten, L., F. Maerten, and P. Gillespie, 2002, Poly3d: a 3d geomechanical program used to solve fault related problems in petroleum reservoirs: Stavanger, Norway, October 16-18 2000.

Meila, M. and J. Shi, 2000, Learning segmentation by random walks: Learning segmentation by random walks:, NIPS, 873–879.

Meila, M. and J. Shi, 2001, A random walks view of spectral segmentation: A random walks view of spectral segmentation:, AI and STATISTICS (AISTATS) 2001.

Pollard, D. Structural geology and geomechanics:. Internet, 2001. http://pangea.stanford.edu/geomech/Research/Research.htm.

Shi, J. and J. Malik, 2000, Normalized cuts and image segmentation: Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence, **22**, no. 8, 838–905.

Stark, T. J., 2004, Relative geologic time (age) volume - Relating every seismic sample to a geologically reasonable horizon: The Leading Edge, **23**, no. 9, 928–932.

Strang, G., 1986, Introduction to applied mathematics: Wellesley-Cambridge Press.

Wheeler, H. E. ., 1958, Time stratigraphy: Bulletin of the american assocation of petroleum geologists, **42**, no. 5, 1047–1063.

Zeng, H., M. M. Backus, K. T. Barrow, and N. Tyler, 1998, Stratal slicing, part I: realistic 3-D seismic model: Geophysics, **63**, no. 2, 502–513.