Surfaces of Minimal Paths from Topological Structures and Applications to 3D Object Segmentation

Dissertation by

Marei Algarni

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

King Abdullah University of Science and Technology

Thuwal, Kingdom of Saudi Arabia

August, 2017

EXAMINATION COMMITTEE PAGE

The dissertation of Marei Algarni is approved by the examination committee

Committee Chairperson: Ganesh Sundaramoorthi
Committee Members: Anthony Yezzi, Wolfgang Heidrich, Peter Wonka

ABSTRACT

Surfaces of Minimal Paths from Topological Structures and Applications to 3D Object Segmentation

Marei Algarni

Extracting surfaces, representing boundaries of objects of interest, from volumetric images, has important applications in various scientific domains, from medicine to geology. In this thesis, I introduce novel mathematical, computational, and algorithmic machinery for extraction of sheet-like surfaces (with boundary), whose boundary is unknown a-priori, a particularly important case in applications that has no convenient methods. This case of a surface with boundaries has applications in extracting faults (among other geological structures) from seismic images in geological applications. Another application domain is in the extraction of structures in the lung from computed tomography (CT) images. Although many methods have been developed in computer vision for extraction of surfaces, including level sets, convex optimization approaches, and graph cut methods, none of these methods appear to be applicable to the case of surfaces with boundary.

The novel methods for surface extraction, derived in this thesis, are built on the theory of Minimal Paths, which has been used primarily to extract curves in noisy or corrupted images and have had wide applicability in 2D computer vision. This thesis extends such methods to surfaces, and it is based on novel observations that surfaces can be determined by extracting topological structures from the solution of the eikonal partial differential equation (PDE), which is the basis of Minimal Path theory. Although topological structures are known to be difficult to extract from images, which are both noisy and discrete, this thesis builds robust methods based on

Morse theory and computational topology to address such issues. The algorithms have run-time complexity $O(NlogN)$, less complex than existing approaches. The thesis details the algorithms, theory, and shows an extensive experimental evaluation on seismic images and medical images. Experiments show out-performance in accuracy, computational speed, and user convenience compared with related state-of-the-art methods. Lastly, the thesis shows the methodology developed for the particular case of surfaces with boundary extends to surfaces without boundary and also surfaces with different topologies, such as cylindrical surfaces, both important cases for many applications in medical image analysis.

ACKNOWLEDGEMENTS

My sincere thanks and appreciation go to my advisor Professor Ganesh Sundaramoorthi for his continuous support and his valuable guidance while working on my Ph.D. His style of scientific research focusing on high quality and important research has changed the way I view research.

Also, I want to thank Professor Anthony Yezzi, Professor Wolfgang Heidrich, and Professor Peter Wonka for being a part of the thesis committee and for their insightful comments and encouraging feedback.

Also, I am grateful to my parents, my wife and my two children for their support and inspiration. They were always supporting and encouraging me.

Additionally, I am grateful for the funding sources, Saudi Aramco, that allowed me to pursue my graduate study and provided me with this opportunity.

From Saudi Aramco, I want to thank Mohammed Al Hamad, Maan Al Hawi, and Nasher Ben Hasan for their cooperation and support during my career at Saudi Aramco and also during my internship. I am fortunate to work with such amazing people.

TABLE OF CONTENTS

Examination Committee Page **2**

Copyright **3**

Abstract **4**

Acknowledgements **6**

List of Figures **10**

List of Tables **16**

1 Introduction **17**
1.1 Contributions of the Thesis 18
1.2 Structure of the thesis 19
1.3 Related Work 19
1.3.1 Crease Surfaces Extraction 19
1.3.2 Level Set Methods 21
1.3.3 Minimal Path Methods 25
1.3.4 Attempts at Surface Extraction by Minimal Paths 29
1.3.5 Graph Cut Methods 30
1.3.6 Minimal Surface Approaches 33
1.4 Applications 35

2 Mathematical Preliminaries **39**
2.1 Introduction and Motivation 39
2.2 Basic Topology 40
2.2.1 Topological Spaces and Continuous Functions 40
2.3 Discrete Topology 44
2.3.1 Cubical Complexes and the Collapse Operation 44
2.4 Manifolds and Calculus of Functions on Manifolds 47
2.4.1 Manifolds 47

2.4.2 Calculus on Manifolds 49
2.4.3 Critical Structures 51
2.5 Morse Theory 54
2.5.1 Morse Functions 55
2.5.2 Morse Complex 57
2.6 Summary and Outlook 59

3 Surface Extraction: Theory and Algorithms 60
3.1 Overview of Method 61
3.2 Surface Boundary Extraction 61
3.2.1 Fronts Localized to the Surface With Fast Marching 62
3.2.2 Contours on the Surface from Front 1D Maxima 63
3.2.3 Maximal Curve Extraction Using the Morse Complex 67
3.2.4 Stopping Criteria and Surface Boundary Extraction 75
3.3 Surface Extraction 78
3.3.1 Surface Extraction Algorithm and Rationale 79
3.3.2 Generalized Minima: Surface of Minimal Paths 82
3.4 The Case of Intersecting Surfaces 85
3.5 Generalizing Surface Extraction to Other Topologies 87
3.5.1 Closed Surface Extraction 88
3.5.2 Cylindrical Topology Surface Extraction 90
3.6 Summary and Outlook 90

4 Implementation Details of Topological Operations 92
4.1 Cubical Complex Storage and Access 92
4.1.1 Data Structures 93
4.1.2 Constructing the Cubical Complexes 98
4.2 Collapse Operation Implementation 100
4.3 Topological Structure Extraction Implementation 102
4.3.1 Morse Complex Extraction Implementation 102
4.3.2 Morse Complex Simplification Implementation 104
4.4 Generalized Minima Surface Extraction Implementation 109

5 Experiments and Evaluation 111
5.1 Datasets and Parameters 111
5.2 Evaluation Methodology 113
5.3 Evaluation 114

5.3.1 Synthetic Data: Surface Extraction Given Boundary 114
5.3.2 Seismic Data: Surface and Boundary Extraction 115
5.3.3 Lung CT Data: Surface and Boundary Extraction 123

6 Conclusion and Future Work **126**
6.1 Conclusion 126
6.2 Future Work 128

References **131**

Appendices **140**

LIST OF FIGURES

1.1 Example graph construct with source and sink. Cut is green dashed line. 31
1.2 [Left]: Schematic of minimal surface formulation. [Right]: Fault surface extraction result using LP 35
1.3 Four interpreted seismic horizons shown (horizontal surfaces). From top to bottom, green, blue, red, orange 37
1.4 Other example application [Left]: Left ventricle surface extraction from CT scan. [Right]: Knee cartilage surface extraction from MRI 38

2.1 [Left two images]: Illustration of faces that form a cubical complex (left) and faces that do not form a cubical complex (0,1,2-faces are marked in red, green and orange). The missing 1-face and 0-faces circled in blue on the right are not in the complex, but they are subfaces of other faces in the set. [Right two images]: Example of 1-face, 0-face free pairs, and 2-face, 1-face free pairs (circled in blue). 46
2.2 Morse Lemma (2D case): level sets in a neighborhood of a point are related by a change of coordinates to one of these forms. 56
2.3 [Left]: Ascending and descending manifolds of a one-dimensional function. [Right]: Descending manifolds of a two-dimensional function (each color represents a separate descending manifold). 58

3.1 Overview of SurfCut. Starting from a user specified seed point on the surface, a front is propagated (left), curves are extracted (middle left), a cut of these curves is performed forming the boundary (middle right), and the surface is extracted (right). 62
3.2 Fast march travel in 2D [Left]: Fast marching fronts (green). [Right]: Minimal paths from a front (yellow). It shows that the local maxima of Euclidean length of minimal paths on front lies on curve of interest 64

3.3 [Top left]: The evolving Fast Marching (FM) front at two different time instances in orange and white. The function $1/\phi$ evaluated at x is the likelihood of surface passing through x, and is visualized (red - high values, and blue - low values). The fronts are localized near the surface of interest. Ridge points of U_E, the Euclidean path length of minimal weighted paths, lie on the surface of interest. [Top right]: This is more easily seen in 2D where the local maxima of the Euclidean path length (red balls) of minimal paths (dashed) are seen to lie on the curve of interest. The green contour is a snapshot of the front. [Bottom]: Schematic in 3D with a front (blue), surface (green), and several minimal paths (orange). Orthogonal to the surface where the surface intersects the front, the Euclidean path length decreases. Along the surface, the path lengths may increase or decrease. This indicates a one-dimensional generalized maxima. 65

3.4 Schematic of quantities in the proof of Proposition 1. 66

3.5 Structure of interest and relation to the Morse Complex. Boundaries between ascending manifolds form one dimensional generalized maxima, i.e., maximal curves. [Left]: There are two ascending manifolds $A(p_1)$ and $A(p_2)$ and the boundary between them $\partial A(p_1)$. If ones follows the gradient descent path from any point on the interior of $A(p_1)$ one will go to a unique minima. Also, following the gradient descent on all points on the interior of $A(p_2)$ will lead to a minima on the outside (not shown). [Right]: It can be noticed that the boundary between ascending manifolds meets where the gradient in the direction normal to the ascending manifold boundary is zero. Also, on either side close to the boundary, the gradients in the N_1 direction are increasing towards the maximal curve. Thus, this confirms that such a curve is a one-dimensional generalized maxima. These one-dimensional generalized maxima in the form of curves are what we seek to be on the surface of interest. 69

3.6 [1st image]: Front color coded with Euclidean path length U_E (top view). Red indicates high values. The bottom view (not shown) is a symmetric flip. Topologically, U_E forms a volcano structure (maximal curve, i.e., top of volcano, is darkest red), and inside the volcano is blue. [Subsequent images]: Illustration of iterations (from left to right) of Algorithm 2 on noise-less data to obtain the ridge curve (white) on the Fast Marching front (green) by computing the Morse complex of U_E. The maximal curve lies on the surface of interest (red). 73
3.7 Illustration of Algorithm 3 operating on noisy data to obtain the highest ridge. 73
3.8 Real data example of Algorithm 3 operating on noisy data to obtain the highest maximal curve. In this example, it required three iterations to converge. 74
3.9 [Left]: Maximal curve (white) extraction by retracting the Fast Marching front at two instants. [Right]: An example cut (red) of ridge curves, forming the surface boundary. Notice that the cut matches with the end of high $1/\phi$ (bright areas). 74
3.10 Visual verification that the maximal curves are on the surface. The maximal curves are shown along with the surface extracted from the algorithm in the next section, which uses only the boundary curve and not the maximal curves shown. 74
3.11 Graph formulation for boundary curve extraction from maximal curves. [Left]: Maximal curves that form the graph for the graph cut algorithm. [Right]: Graph in which all maximal curves are resampled to have the same number of nodes. 76
3.12 Quantities defined in the proof of Proposition 3. 81
3.13 Schematic diagram to illustrate surface extraction algorithm defined in the continuum, and the associated quantities in the algorithm definition (see text). 81
3.14 Illustration of valley extraction by Algorithm 4, which retracts the volume while preserving 1-faces on the surface boundary (red). This gives the surface of interest. 83

3.15 Synthetic example of extracting a sphere with top cut such that the boundary is four arcs. The image (not shown) is a noisy image of the cut sphere with holes. Maximal curves are extracted via Algorithm 1 (top left). The final cut of maximal curves (top, middle left), the final surface extracted via Algorithm 2 (top, middle right), and the ground truth (top, right) are shown. Snapshots in the removal of faces in Alg. 2 are shown (bottom), resulting in the surface (right). 83
3.16 Surface intersection in lung data. Right lung contains, one surface. Left lung contains, two intersecting surfaces that require the surface intersection handling. 86
3.17 Closed surface extraction. [Left]: Input a 3D image containing closed surface and a seed point picked by the user anywhere on the object of interest. [Middle]: Fast Marching algorithm is computed from seed point $U(p) = 0$. [Right]: Extracted closed surface obtained by extracting boundaries of descending manifolds. 89
3.18 Visualization of the iterations of the algorithm for extraction of boundaries of descending manifolds. Since the data are noise, this example required four iterations to converge. Note that the closed surface is cut from the middle for the purpose of visualization to see how the shape is simplified at each iteration of the algorithm. 89
3.19 Cylindrical topology extraction. [Left]: Input a 3D image containing cylindrical surface and closed curves on each ending slice. [Middle and Right]: Extracted cylindrical topology (extract boundaries of descending manifolds). 91

4.1 Cubical complex elements in 3D. 93
4.2 Edges and faces index types in 3D images [Left]: Edges index types. [Right]: Faces index types. 94
4.3 [Left]: Cubical complex lattice in 3D. [Right]: Example showing indexing of nodes, edges and faces in 2D. A unique number is specified for each node, edge and face. 96
4.4 Cases encountered when performing the collapse operation for Morse complex extraction. [Left]: The edge is a minima. [Middle]: A free edge. [Right]: An isthmus (non-free) edge. 102

4.5 Example highest maximal curves extraction of cubical complex (iterative Morse complex simplification). [Left]: Cubical complex in 2D. [Middle]: Maximal curves after first iteration. [Right]: Highest maximal curves in final iteration, and final simplfied complex consisting of two "super-faces" and one "super-edge". 109
4.6 Generalized minima of a function are generalized maxima of the negative function. [Left]: Generalized 1-dim minima. [Right]: Generalized 1-dim maxima. 110

5.1 Example surfaces in the synthetic dataset. Each surface has a different boundary curve, and the surfaces are of different shape, exhibiting various degrees of randomness. 112
5.2 **Quantitative Analysis of Smoothing Parameter** Boundary (left) and surface (right) F-measure versus smoothing degradations for my method and [1]. 116
5.3 **Qualitative Analysis of Smoothing Parameter**. Results displayed by varying the parameter in ϕ (larger towards the right). Surfaces extracted by Crease surfaces and my method are displayed with the ground truth. 117
5.4 **Quantitative Analysis of Noise Degradations** Boundary (left) and surface (right) F-measure versus the noise degradation plots for my method and [1]. 118
5.5 **Qualitative Analysis of Noise Degradations**. Results displayed by varying the additive noise to ϕ (larger towards the right). Surfaces extracted by Crease surfaces and my method are displayed with the ground truth. 118
5.6 **Slice-wise Validation on Seismic Data**: [Left column]: Slices corresponding to x-y, x-z, and y-z planes, [Middle, left]: Local surface likelihood ($1/\phi$), [Middle right]: Intersection of SurfCut result (green) with slice, [Right column]: surface from SurfCut at certain viewpoint. 119
5.7 **Robustness to Seed Point Choice**: [Left]: A visualization of the seed points chosen. [Right]: Boundary F-measure versus various seed point indices. The same boundary and surface accuracy is maintained no matter the seed point location. 120

5.8 **Quantitative Analysis of Sensitivity of Threshold** This analyzes the sensitivity of the boundary curve extraction to the threshold on the average cut cost to stop the algorithm. The accuracy of the boundary and surface measured by F-measure versus various thresholds is plotted. 121

5.9 Example result in an automated setting. [Left]: Result by Crease Surfaces, which contains holes and incorrectly detects clutter (top, red) due to noise in the data. [Right]: Results of SurfCut, which extracts the correct number of surfaces and produces smooth simple surfaces. 122

5.10 **Slice-wise Validation in Lung CT Dataset**. [Top]: Various slices of an image of a patient, [Bottom]: Surface generated with SurfCut intersected with the slice above (green) superimposed on the slice. Notice the structure of interest is a subtle thin lines in the slices (top). . . . 124

5.11 **Qualitative Results on Lung CT**. Columns show the surfaces on the same slice on the same patient for various methods. Moving through a row shows the surface for different patients and a slice of the image is shown for various different slices. SurfCut extracts more of the fine structure of the fissures, better estimates the boundary and recovers more of the surfaces than Crease surfaces. 125

LIST OF TABLES

4.1 Summary of data structures for storing and processing the cubical complex. 98

5.1 Comparison of methods for surface extraction given the surface boundary on the synthetic dataset. Speed (in seconds), surface precision (P), recall (R), F-measure (F), and ground truth covering (GT-cov) are reported. Higher P, R, F, GT-Cov. indicate better fidelity to the ground truth. 115

5.2 **Quantitative Evaluation on Lung Dataset**. Comparison of methods in terms of surface and boundary accuracy. Precision (P), recall (R), F-measure (F), and ground truth covering (GT-cov) are reported. Higher P, R, F, GT-Cov. indicate better fidelity to the ground truth. 124

Chapter 1

Introduction

Minimal path methods [2], built on the Fast Marching algorithm [3] (see also [4]), have been widely used in computer vision. They provide a framework for extracting continuous curves from possibly noisy images. For instance, they have been used in edge detection [5] and object boundary detection [6], mainly in interactive settings as they typically require user-defined seed points. Because of their ability to provide continuous curves, robust to clutter and noise in the image, generalizations of these techniques to extract the equivalent of edges in 3D images, which form surfaces, have been attempted [7, 8]. These methods apply to extracting a surface with a boundary that forms a curve, possibly in 3D, which is called a *free-boundary*. Extraction of surfaces with free-boundary is important because many edges form these surfaces, and edges are fundamental structures that are prevalent in images. Some applications include medical datasets (e.g., lung fissures, walls of heart ventricles) [9] and scientific imaging datasets (e.g., fault surfaces in seismic images, an important problem in the oil industry) [10]. In [9] an alternative method to extract such surfaces, based on the theory of minimal surfaces [11], is provided. However, existing approaches to surface extraction for surfaces with free-boundary have a limitation - they require the user to provide the boundary of the surface or other users laborious input.

In this thesis, Fast Marching algorithm and techniques from computational topology are used to create an algorithm for extracting the boundary of a surface from a 3D image and a single seed point, and an algorithm to extract the surface. The main idea is to use Fast Marching to "smooth" a local (possibly noisy) likelihood map of

the surface in a way that is guaranteed to preserve locations of critical structures, and then extract the structures with methods, built from computational topology, that guarantees correct topology. I show that the resulting structures correspond to the surface of interest, and the surface is a collection of minimal paths. The method is applicable to any imaging modality, and can be used to extract any simple surface with a boundary from an image that contains noisy local measurements (possibly an edge map) of the surface. I demonstrate the method on two applications - fault extraction from seismic images, and lung fissure extraction from CT.

1.1 Contributions of the Thesis

The contributions are: **1.** I introduce the first algorithm, to the best of my knowledge, to extract a closed 3D space curve forming the boundary of a surface from a single seed point. It is based on extracting critical structures from a distance produced by Fast Marching. **2.** I introduce a new algorithm, based on extracting a critical structure of the FM distance, to extract a surface given its boundary and a noisy image. It produces a topologically simple surface whose boundary is the given space curve. The surface is shown to be formed from minimal paths. Both boundary and surface extraction have $O(N \log N)$ complexity, where N is the number of pixels. **3.** I provide a fully automated algorithm using the algorithms described below to extract all such surfaces from a 3D image. **4.** I show that the methodology developed for surfaces with boundaries can be more generally applied to other surface topologies, such as surfaces topologically equivalent to a sphere and cylindrical surfaces. **5.** The method has been tested on challenging datasets, and I quantitatively out-perform comparable state-of-the-art in free-boundary surface extraction.

1.2 Structure of the thesis

In the rest of this chapter (**Chapter 1**), I begin by reviewing the work in literature and related it to the method. Level sets, minimal paths, Graph Cuts, and crease surface methods are reviewed. Then, I list some applications of free-boundary surface extraction in the domain of seismic and medical imaging. In **Chapter 2**, computational topology concepts that are used in this thesis such as cubical complex theory, deformation retracts, and its counterpart collapse operation in the discrete domain are introduced. I also introduce the basics of Morse theory that are used in the design of my algorithms. In **Chapter 3**, I will introduce the algorithm to extract a free boundary surface. This includes both extracting surface boundary curve from a seed point and a free-boundary surface given the boundary curve. I will also show how these algorithms can be extended to close surfaces and cylindrical surfaces. In **Chapter 4**, I will introduce an efficient implementation of the algorithms. In **Chapter 5**, I will detail the experiments of the synthetic dataset, seismic dataset, and lung CT scan dataset. This includes performance and accuracy analysis.

1.3 Related Work

Five bodies of literature for surface extraction, including Crease Surface Extraction, Level Set Methods, Graph Cut Methods, Minimal Surfaces, and finally Minimal Path Methods are reviewed. The method builds on the last.

1.3.1 Crease Surfaces Extraction

Perhaps the simplest method for surface extraction, and open surface extraction, in particular, is a method called Crease Surfaces [1]. The method requires as input the local the likelihood that a pixel belongs to the surface, which is ideally 0 away from the surface and 1 on the surface. Since in applications, where the original image can

be noisy, the likelihood is typically also noisy, and to cope with this smoothing of the likelihood map is performed and a local differential operator (based on the smoothed Hessian matrix) is used to obtain a refined likelihood map that mitigates noise.

The original likelihood map ϕ is smoothed (for example with a Gaussian), and the Hessian matrix H, consisting of combinations of second order partial derivatives are computed. It can then be decomposed into an eigen-decomposition as

$$H = \lambda_{\mathrm{u}} u u^{\mathrm{T}} + \lambda_{\mathrm{v}} v v^{\mathrm{T}} + \lambda_{\mathrm{w}} w w^{\mathrm{T}}, \tag{1.1}$$

where the eignevalues are ordered as

$$\lambda_{\mathrm{u}} \geq \lambda_{\mathrm{v}} \geq \lambda_{\mathrm{w}}, \tag{1.2}$$

and u, v, w are the corresponding eigenvectors. The local structure of the image and/or ϕ can be inferred from their Hessian decomposition, including linear, blob or ridge structures. For determining pixels that lie on the surface, the smallest eigenvalue λ_{w} is a large negative, compared to two small positive eigenvalues. If true, the eigenvector w should be orthogonal to the surface. The refined likelihood map can then be formulated to be large when λ_w is a large negative, and the others are small negatives.

As in [1] and [12], h vector is computed as follow:

$$h = (1 - \lambda) w w^T g \tag{1.3}$$

where

$$\lambda = \begin{cases} 0, & \lambda_v - \lambda_w > \varepsilon \\ (1 - \frac{\lambda_v - \lambda_w}{\varepsilon})^2 & otherwise, \end{cases} \tag{1.4}$$

Once the refined likelihood map is computed, connected components of the maxima of this likelihood are computed, obtaining several surfaces within the image. This method is convenient since it is fully automated. However, it processes raw data without incorporating geometrical priors on realistic surfaces, it is sensitive to noise, produces undesirable holes in the surface, and the surface is highly inaccurate, especially for seismic images, as I show in experiments. This approach has been applied to seismic images for extracting fault surfaces [12], and despite its problems it is regarded as the state-of-the-art in that field.

1.3.2 Level Set Methods

There are methods in computer vision for incorporating geometrical priors so that the surface extracted is guaranteed to be, for example, smooth, have no holes, and other geometrical properties. A large body of literature [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28] for extracting surfaces with geometrical consistency is *Level Set Methods*, primarily emanating from the seminal paper [29]. Although these methods are not directly used in this approach, they are important historical information for the methods that I introduced.

Level set methods trace their history in computer vision from active contours, or snakes, first introduced by Kass et al. [30]. The idea is to evolve an initial closed curve or closed surface to some features in the image like edges to form the boundary of an object while keeping the curve or surface smooth, as object boundaries are typically smooth. Without incorporating smoothness, the curve could get stuck in fine-scale features of the image, such as noise that is typical in images. The evolution of the curve is determined by formulating an energy minimization problem on curves or surfaces. The energy defined in active contours is

$$E(C) = \alpha \int_0^1 |C^{'}(q)|^2 dq + \beta \int_0^1 |C^{''}(q)|^2 dq + \lambda \int_0^1 g(C(q)) dq, \qquad (1.5)$$

where $C : [0,1] \to \mathbb{R}^2$ is a parametric representation of a curve with the constraint that $C(0) = C(1)$ so that the curve is closed, q is the parameter of the curve, $g : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ is the edge stopping function, C' and C'' denote the first and second derivatives of the curve, and $\alpha, \beta, \lambda > 0$ are parameters that must be chosen. Note the edge stopping fucntion is chosen as $g(x) = 1/(1+|\nabla I(x)|)$, where ∇I is the gradient of the image I. The idea is that the energy is small when the curve is aligned to the edges due to the third term, and the first two terms are small when the curve is smooth, as the derivatives are small when the curve is smooth. The minimization of the energy is performed by a mathematical technique called the *Calculus of Variations*, which is a generalization of optimization techniques in finite dimensions, to infinite dimensions. The optimization works by starting with an initial guess of the curve (either hand drawn or performed in come semi-automated way), and the curve is driven to minimize the energy by following the gradient of E, determined by calculus of variations. The minimization is then a partial differential equation (PDE):

$$\frac{\partial}{\partial t} C(p,t) = \alpha C_{pp}(p,t) - \beta C_{pppp}(p,t) - \nabla g(C(p,t)), \tag{1.6}$$

where t is an artificial time parameter that parametrizes the evolution of the curve.

The original active contours model, however, has several drawbacks, including that it is difficult to implement in a numerically stable fashion. Indeed to implement it, the curve must be discretized and represented by a finite set of particles, and this presents problems, for example, a resampling strategy must be used to cope with growth and shrink of the curve. Also, it is possible that due to image noise the curve may self-intersect, and this must be avoided to prevent instabilities. *Level set methods* [29] have been derived to avoid all of these complications with representing a curve with particles, and provides a numerically convenient way to implement a curve evolution. Level set methods are *geometric* methods in the sense they do not depend

on a particular way of parameterizing the curve and do not keep track of particles. As such, they are typically used to implement evolutions that are geometric. A geometric evolution similar to the original active contours model is known as Geodesic active contours and was introduced by [31] to make use of level set methods. To derive a geometric evolution, a geometric energy (depending only on the geometry of the curve and not the way the curve is parameterized) is considered. The energy functional is given by

$$E(C) = \int_C g(C(s))ds, \tag{1.7}$$

where s denotes the arc-length, $\mathrm{d}s$ is the arc-length measure, and the curve C is parametrized with arclength parameterization. By using arc-length, the energy is geometric. The term, *geodesic*, comes from the fact that a geodesic is a curve that minimizes length, and the approach minimizes a weighted length. Note the derivative terms in the original active contour model no longer appear. This is because by incorporating the arc-length measure, length of the curve is minimized while keeping the curve close to image edges. Since length is minimized, the curve becomes smooth as non-smooth curves typically have large length. The curve evolution to minimize the energy is given by

$$\frac{\partial}{\partial t}C = -(\nabla g \cdot N)N + \kappa g N, \tag{1.8}$$

where N is the inward normal vector to N, and κ is the curvature of the curve. The first term drives the curve to image edges and the second term smooths the curve while slowing down the smoothing near image edges.

The geometric curve evolution for Geodesic active contours was implemented using Level Set Methods so that the numerical difficulties with the original active contours model could be addressed. Level Set Methods introduced in [29] evolve curves or surfaces by implicitly representing them as the zero-level set of a function defined on the image domain Ω. For curves, the level set function is a function $\Psi : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$

(and for surfaces, $\mathbb{R}^2$ is replaced by $\mathbb{R}^3$). Thus, the curve evolution is transformed to an evolution of a function in two dimensions. The level set function maps the image plane at any time t to the space of real numbers, $\Psi : \mathbb{R}^+ \times \mathbb{R}^2 \to \mathbb{R}$. The zero level set of Ψ is set to be the curve C at each time t:

$$\Psi(t, C(t, p)) = 0 \qquad \forall p \in [0, 1] \tag{1.9}$$

Given that the curve evolves according to

$$\frac{\partial}{\partial t} C(p, t) = F(p, t), \tag{1.10}$$

where F is some function, the evolution of the level set function can be obtained by computing the time derivative of (1.9), which gives

$$\frac{\partial}{\partial t} \Psi(t, x) = -\nabla \Psi(t, x) \cdot F(p, t) \tag{1.11}$$

where $x = C(p, t)$ is on the curve, and ∇ denotes the spatial derivative. Note that one must extend F to points outside of the curve. For geodesic active contours, the level set evolution is

$$\frac{\partial}{\partial t} \Psi(t, x) = \nabla g(x) \cdot \nabla \Psi(t, x) + g(x) |\nabla \Psi(t, x)| \mathrm{div} \left(\frac{\nabla \Psi(t, x)}{|\nabla \Psi(t, x)|} \right). \tag{1.12}$$

The evolution of the curve is thus converted to an evolution of a function defined on the entire image. The advantage of this approach is that resampling strategies as used in particle implementations are not needed, and the method can naturally handle splitting and merging of curves naturally without any additional effort. Further, they can be implemented in computationally efficient ways so that the cost is nearly the same as particle-based methods.

Although the Geodesic active contour implemented with level set methods is more numerically convenient than original active contours, active contours require an initialization, and in general, such methods are sensitive to the initialization. This is because they are based on minimization of energies that are non-convex and are optimized with local optimization methods. While advances in level set based methods have come for certain energies by relaxing them to a convex optimization problem, and then applying convex optimization techniques, they are not applicable to extraction of open surfaces using the geodesic active contour energy, and in fact neither are level set methods.

1.3.3 Minimal Path Methods

The method for surface extraction is built on the theory of *minimal paths* [4, 2]. This theory gives a *globally* optimal method for optimizing the energy for Geodesic active contours (1.7) for the case of open contours, not open surfaces. The method will show how the theory can also be applied to extract open surfaces, but first, let us go through the original theory.

Let $S \subset \Omega \subset \mathbb{R}^n$ $(n \leq 2)$ be a closed set, which can, for example, be a point, a closed curve or a closed surface in $\mathbb{R}^n$ (Ω is the image domain). I consider the following problem that generalizes the geodesic active contour energy to $\mathbb{R}^n$ for curves with one endpoint q fixed and the other lying in S:

$$\inf_{\gamma \in \Gamma_q} \int_0^1 \phi(\gamma(t)) \, |\gamma'(t)| \, dt := \inf_{\gamma \in \Gamma_q} L_\phi(\gamma) \tag{1.13}$$

where $\phi : \Omega \to \mathbb{R}^+$ is a local cost, $\gamma : [0,1] \to \Omega$ is an open curve that belongs to the set of curves defined by

$$\Gamma_q = \{\gamma : [0,1] \to \Omega \, : \, \gamma(0) = q, \gamma(1) \in S\} \, . \tag{1.14}$$

Notice that $ds_\gamma = |\gamma'(t)|\,dt$ is the arclength element of the pathγ, and thus

$$L_\phi(\gamma) = \int_0^1 \phi(\gamma(t))\,|\gamma'(t)|\,dt \tag{1.15}$$

is a weighted length of γ (the weight ϕ being spatially dependent). Thus, it can be seen that (1.13) is the weighted length of the smallest weighted length path starting from q and ending at any point on S. Minimal path theory determines such a path of *globally* minimal weighted path length.

It can be shown that the globally optimal path from q (or even *any* point in Ω) can be determined by solving the eikonal partial differential equation:

$$\begin{cases} |\nabla U(x)| = \phi(x) & x \in \Omega \backslash S \\ U(x) = 0 & x \in S \end{cases}. \tag{1.16}$$

Indeed to compute the globally minimal weighted path length from a point q, one can follow the negative gradient to the set S by solving the following ordinary differential equation (ODE):

$$\begin{cases} \gamma'(t) = -\nabla U(\gamma(t)) & t > 0 \\ \gamma(0) = q \end{cases}. \tag{1.17}$$

The path traced out by the above ODE is guaranteed to end at some point in S, and is guaranteed to be the globally minimal weighted path. Note one can choose q to be any point and trace out the path using the same U to find the minimal path from any other point. The proof of this fact can be shown by first showing that the path defined by (1.17) satisfies the Euler-Lagrange equation for L_ϕ, which is given by

$$\nabla L_\phi(\gamma) = \nabla\phi(\gamma(s)) - \frac{d}{ds}(\phi(\gamma(s))\gamma_s(s)) = 0, \tag{1.18}$$

where s is the arc-length parameter, and the sub-script denotes the derivative. So-

lutions to the Euler-Lagrange equation is known to be a local optimizer. With an additional argument based on the Principle of Dynamic Programming, one can show that this local optimizer is in fact the global optimizer. The solution of the eikonal equation U evalutated at a point q gives the weighted length of the minimal path from q to S. For this reason, U is referred to as a *distance* function.

A fast numerical method to solve the eikonal equation for U is known as *Fast Marching Method* [32], and described next. It can be shown that the solution of the eikonal equation is equivalent to evolving the set S in its outward normal direction with a speed $1/\phi$, i.e.,

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial t} = \frac{1}{\phi} N & t > 0 \\ \mathcal{L}(0) = S & t = 0 \end{cases}. \tag{1.19}$$

At each point x, the distance U at x is the time t that the front $\mathcal{L}$ *arrives* at x. This fact gives an efficient scheme to compute the solution of U and is the basis for the Fast Marching algorithm. Indeed, the idea of the algorithm is simply to evolve the surface S (propagating the level sets of U outward) and simultaneously recording the arrival times t at the each of the points of $\mathcal{L}(t, .)$ using a discretization of the eikonal equation (3.1).

Specifically, the Fast Marching algorithm starts by initializing the distance to zero at points on S and infinite elsewhere. The algorithm updates an array of labels that marks points as ALIVE, TRIAL or FAR. FAR points have not been visited, TRIAL points that are to be visited, and ALIVE points are visited points. The points on S are initially marked as TRIAL and all other points are marked as FAR. All TRIAL points are added to a heap (ordered so that the minimum value is on the top of the heap). At each iteration, the point at the top of the heap is removed and labeled as ALIVE. The neighbors of the removed point are then added to the heap if they are marked as ALIVE. Those points have values determined by solving a discretization of the eikonal equation using current values of distances. Any neighbor that is marked

as ALIVE has its distance value updated by resolving the eikonal equation at the point. The method works in any dimensions. The algorithm pseudocode is given below:

Algorithm 1 Fast Marching

1: lij: label of the pixel ij
2: Initialize, $U_{ij} = 0 \in S$, l_{ij}=TRIAL $\in S$
3: Initialize $U_{ij} = \infty \notin S$, l_{ij}=FAR $\notin S$
4: **repeat**
5: $i_m, j_m = argmin_{ij \in TRIAL} U_{ij}$
6: $l_{i_m, j_m} = ALIVE$
7: **for** neighbor kl $\in i_m, j_m$ **do**
8: **if** l_{kl} = FAR **then**
9: l_{kl} = TRIAL
10: **end if**
11: **if** l_{kl} = ALIVE **then**
12: $(max(u - U_{i-1,j}, u - U_{i+1,j}, 0))2 + (max(u - U_{i,j-1}, u - U_{i,j+1}, 0))^2 = \phi_{ij}$
13: $set\, U_{kl} = min(u, U_{kl})$
14: **end if**
15: **end for**
16: **until** all grid points have been marked ALIVE

The Fast Marching method is known to have better accuracy than discrete algorithms based on Dijkstra's algorithm [33]. The computation of minimal paths based on Fast Marching has been used in 2D images to compute edges of images, guaranteeing that the edge does not have gaps and is smooth. This is especially useful as edges traditionally computed using differential operators are noisy, and produce gaps. A limitation of the minimal path approach is that it requires the user to input two points - the initial and ending point of the curve. In [5], endpoints are successively detected automatically based on extrema of the Euclidean length of minimal paths, but the algorithm's stopping criteria to determine the endpoint of the curve or edge of interest is sensitive to noise and other fine image features. These methods are not directly applicable to extracting a surface forming an edge in 3D, which is of interest

in this thesis.

1.3.4 Attempts at Surface Extraction by Minimal Paths

In [8], minimal paths are used to extract a surface edge topologically equivalent to a cylinder. The user inputs the two boundary curves of the cylinder (that must lie in a 2D plane) and minimal paths joining a point on one curve to the other curve are computed conveniently using the solution of a partial differential equation, which is described next. The collection of all minimal paths from points in the first planar curve to the other is embedded as the zero level set of a function Ψ. Let U to be the distance function from the second planar curve to all points in the volume. Since minimal paths are parallel to ∇U and $\nabla\Psi$ is perpendicular minimal paths as minimal paths are the zero level set of Ψ, the PDE for Ψ is

$$\nabla\Psi(x)\cdot\nabla U(x)=0, \tag{1.20}$$

for points on the zero level set of Ψ. Extending the constraint given by equation (1.20) to the whole domain Ω will give a sufficient condition for determining $\Psi(x)$. To enforce smoothness, regularization is added, which yields the PDE

$$\begin{cases} \nabla\Psi(x)\cdot\nabla U(x)-\alpha\Psi(x)=0 & x\in\Omega-\partial\Sigma \\ \Psi(x)=d_{\partial\Sigma}(x) & x=\partial\Sigma \end{cases}, \tag{1.21}$$

where $\partial\Sigma$ is first planar curve, $d(\partial\Sigma)$ is the sign distance to $\partial\Sigma$, defined on the plane where Σ lies. This is a transport equation that can be solved using finite difference methods. Note the transport equation transports the data $d_{\partial\Sigma}$ along the gradient of U toward the plane of the second curve. While the method deals with the problem of gaps in the surface when naively computing a series of minimal paths by following the gradient of U from the first curve to the second curve, the method is sensitive to

the choice of α, which must be chosen carefully. Further, the method only applies to the special cylindrical topology, which is not sufficient for the applications that are considered in seismic images.

Surface extraction with less intensive user input (i.e., without the two boundary curves) was attempted in [7]. There, a patch of a sheet-like surface is computed with a user-provided seed point and a bounding box, with the assumption that the patch slices the box into two pieces. The algorithm extracts a curve that is the intersection of the surface patch with the bounding box using the distance function to the seed point obtained with Fast Marching. Once this boundary curve is obtained, the patch is computed using [8]. The obvious drawbacks of this method are that only a patch of the desired surface is obtained, and a bounding box, which may be cumbersome to obtain, must be given by the user.

1.3.5 Graph Cut Methods

Another popular approach for determining curves and surfaces in images is *Graph Cuts*. An advantage over traditional level set methods is that some energies may be optimized to give a *globally* optimal solution. Although the method cannot directly deal with open surfaces, which is of interest in this thesis, a review of some of the Graph Cut literature is described next since some of the techniques are used as a sub-routine of the algorithm.

Graph Cuts is a general and powerful discrete optimization tool that is used in many fields since 1960 [34, 35, 36, 37, 38, 39, 40, 41, 42] . It is used widely in computer vision applications such as segmentation, optical flow estimation and much more. Graph cuts can be applied to a discretization of the geodesic active contour energy and can be used to solve a variant of that problem. Problem description and its basic idea for optimization are summarized next. Given a graph $G = (V, E)$, which is a collection of vertices and edges, with edges of the form (u, v) where $u, v \in V$. The

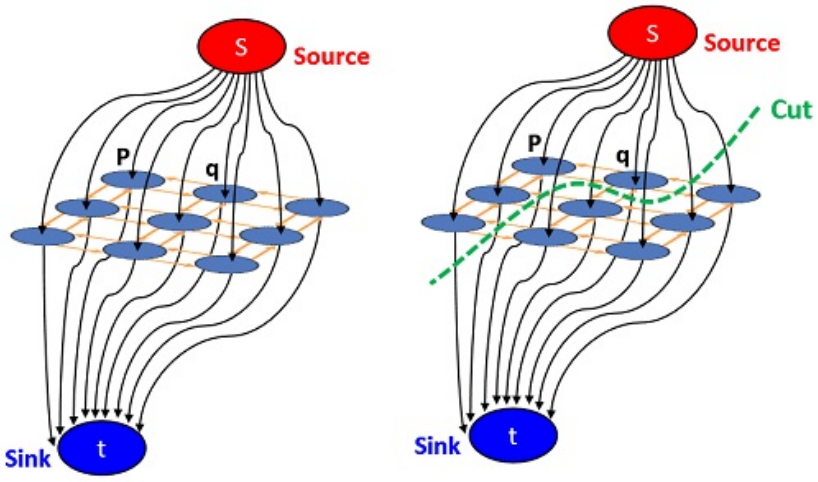

Figure 1.1: Example graph construct with source and sink. Cut is green dashed line.

graph could, for example, be formed from pixels in the image as images and adjacent pixels to a pixel forming the edges. Denote two special vertices s and t, called the *terminal* and *sink* nodes. Define a *cut* to be a collection of edges that divides the graph into two disjoint components, S and T with edges (u, v) in the cut such that $u \in S$ and $v \in T$. Further, it is required that $s \in S$ and $t \in T$ so that the source and terminal nodes are in separate sets. Note that for each edge $(u, v) \in E$, there is a cost $c(u, v)$ associated with the edge. The Graph Cut problem is to find the *minimum cut*, i.e.,

$$\min_{S,T \subset V, S \cap T = \emptyset, s \in S, t \in T} \sum_{(u,v) \in E, u \in S, v \in T} c(u, v). \tag{1.22}$$

That is, the problem is to find disjoint subsets S and T, containing s and t, respectively, with minimum sum of costs along edges joining S and T (See Figure 1.1). Note that a cut is approximately equal to the integral along the contour in geodesic active contours.

The optimization of the problem above can be accomplished by looking at the dual problem, which is achieved by a theorem called *Max-Flow Min-Cut*. The problem is converted to a problem of determining maximum flow in a network, for which there

have been many algorithms developed in the network theory literature. I now define the flow and the max flow problem. The problem assumes the same setup as the Graph Cut problem, i.e., a graph, terminal and source nodes, and costs for edges. A flow f is a function defined on edges and satisfies the following properties:

1. Capacity constraint:

$$0 \leq f(u,v) \leq c(u,v). \tag{1.23}$$

2. Flow Conservation:

$$\forall u \in V \backslash \{s,t\}, \sum_{\{v \in V\,:\,(u,v) \in E\}} f(u,v) = 0. \tag{1.24}$$

3. Skew Symmetry:

$$\forall (u,v) \in E, , \; f(u,v) = -f(v,u). \tag{1.25}$$

The goal of the maximum flow problem is to maximize the amount of flow being distributed from the source node to the terminal node, i.e.,

$$\max_{f \in F} \sum_{\{v \in V\,:\,(s,v) \in E\}} f(s,v), \tag{1.26}$$

where F is the set of flows that satsify the three constraints above. The maximum flow problem relates to the minimum cut problem by the following theorem:

Theorem 1.3.1 (The Max-flow/Min-cut Theorem)**.** *In a flow network* (G,s,t,c) *the following holds:*

$$\max_{f \in F} \sum_{\{v \in V\,:\,(s,v) \in E\}} f(s,v) = \min_{S,T \subset V, S \cap T = \emptyset, s \in S, t \in T} \sum_{(u,v) \in E, u \in S, v \in T} c(u,v). \tag{1.27}$$

The proof can be found in [43].

By converting the min-cut problem to a max-flow problem, one can make use of the wealth of algorithms for the latter problem. One of the most popular and first algorithms to achieve this is Ford-Fulkerson algorithm [44]. It works by finding paths from node s to t with a valid flow and argument path until no more flow is possible. It uses depth-first-search (DFS) to find these paths. When no more flow is available, the minimum cut can be extracted as the set of saturated edges (where the flow equals the edge cost), which partitions the graph. Following this idea, the Boykov-Kolmogorov algorithm [35] improved the search process by storing the search history. This algorithm to solve max-flow is the one that is most widely used algorithm in computer vision.

Another set of network flow algorithms used in computer vision is minimum cost circulation network flow (MCNF) [45, 46, 47, 48, 49, **?**]. These algorithms do not require source or sink. It works by finding negative cost cycles and augmenting them as in max-flow, but it constructs cycles instead of flows from s to t. Extensive study and comparisons of MCNF algorithms can be found in [38].

1.3.6 Minimal Surface Approaches

The original Graph Cut problem does not directly apply to the case of open surfaces (surfaces that do not partition the image into disjoint sets). However, given the 3D boundary of the surface, which is a curve in 3D, one can solve a discrete graph-based generalization of the Geodesic active contours problem to surfaces. The method, known as *minimal surfaces*, was presented first in [11] and later re-introduced in computer vision in [50, 9].

The algorithm determines the surface whose boundary is a given 3D curve c and minimizes the following energy:

$$E(S) = \int \phi(x) dS(x), \quad \text{subject to } \partial S = c, \tag{1.28}$$

where $dS(x)$ denotes the surface area element, and ∂S indicates the boundary of the surface (as illustrated in Figure 1.2). Thus, the problem is to minimize a weighted surface area. Although it is argued that minimal surfaces are more natural extensions of the 2D shortest path problem to 3D than minimal path-based methods for surface extraction, there is no compelling reason for choosing a surface that has the bias to minimal surface area and that prior may or may not be relevant in applications.

One can optimize this energy in a globally optimal manner by discretizing the energy. The minimization problem can be solved as a linear program, and a globally optimal solution can be obtained. Indeed, [11] reformulates the energy minimization in discrete form as

$$\min_z w^T z, \text{ subject to } Bz = r \tag{1.29}$$

where $z \in \mathbb{R}^{\mathrm{n}}$ (n is the number of pixels in the image) is a vector formed from rasterizing the pixelized image domain and indicates whether the pixel is on the surface, $w \in \mathbb{R}^{\mathrm{n}}$ is a weight vector formed from rasterizing the function $\phi(x)$, $B \in \mathbb{R}^{\mathrm{n} \cdot \mathrm{m}}$ is the incidence matrix of faces and edges of the graph formed from image pixels and assuming 6-neighbor connectivity, and $r\mathbb{R}^{\mathrm{m}}$ is the indicator function indicating the boundary curve c. Note that z and r are binary vectors. The problem 34, when the indicator functions are relaxed, is a linear program, and this can be solved by any linear programming (LP) solver.

Sullivan in [11] extended the LP program to be solved by MCNF, as LP solvers tend to be slow for large problems. Grady in [9] used this result for surface extraction from 3D images. While the approach is computationally faster than the LP, it has two drawbacks. First, it requires a guess for the initial surface whose boundary is the given boundary curve, and there is no general algorithm for producing this initialization. In general, the method could result in a different solution for different initializations. Second, since the saturated edge do not result in partitions, it requires extra post-processing to extract the set of edges belonging to the surface, which is a

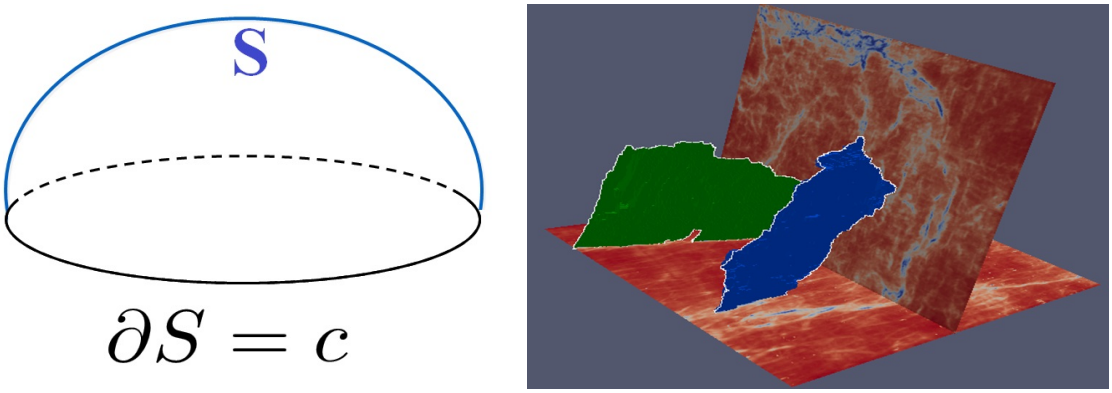

Figure 1.2: [Left]: Schematic of minimal surface formulation. [Right]: Fault surface extraction result using LP

difficult problem in itself and it is time-consuming [51] [52]. Moreover, MCNF is still very slow for practical applications as I will show later in this thesis.

The main drawback of these minimal surface methods is that the user must input the boundary of the surface. Determining the 3D boundary of a surface is a difficult problem in itself, and I am not aware of the existence of a method for this task. Therefore, the user must enter the 3D curve by hand, which is cumbersome and laborious to perform. The method proposed in this thesis generates the boundary automatically, as well as generating the surface. The proposed surface extraction is computationally much quicker than minimal surface approaches as we will see in experiments.

1.4 Applications

An open surface (free-boundary surface) can be depicted as a sheet of paper floating in space. More formally, an open surface is a 2D manifold with boundary that is a 1D curve. Both the surface and the boundary are embedded in 3D space. In various scientific domains, information from the 3D world is captured as a 3D volumetric images. Two major areas of 3D imaging are seismic and medical 3D images. In this thesis, I experimented with one object from each domain. However, there are many

more objects that can be represented as open surfaces. Here, I will mention some of these examples.

3D seismic images are used to investigate the subsurface data for many reasons such as oil and gas exploration, mining and construction. The two major objects that geophysicists and geologists interpret to gain insight of the subsurface data are horizons [53, 54, 55, 56, 57, 58, 59] and faults [60, 10, 61, 62, 60, 63, 64]. Both horizons and faults are modeled as open surfaces. Seismic faults are discontinuities in subsurface data due to significant geological displacement as a result of rock mass movement during plate tectonics, earthquake or other geological processes. Geologists and geophysicists are highly interested in interpreting fault surfaces because it can act as barriers or conduits for oil and gas reservoir. Also, they are important in planning drilling operation and they have a high influence on the mechanical behavior of the rocks around them. In fact, the method was initially motivated by this application due to its importance in oil and gas exploration and the limitation of available methods in the literature. Thus, throughout this thesis and in experiment section, I presented various results for fault surface extraction. Horizons are a boundary that separate two layers in subsurface data due to the difference in lithology (rocks properties and contents). A horizon is shown in Figure 1.3. Geologists and geophysicists need to identify horizons to determine the depth and spatial location of important geological layers such as the layers that contain oil, gas or minerals.

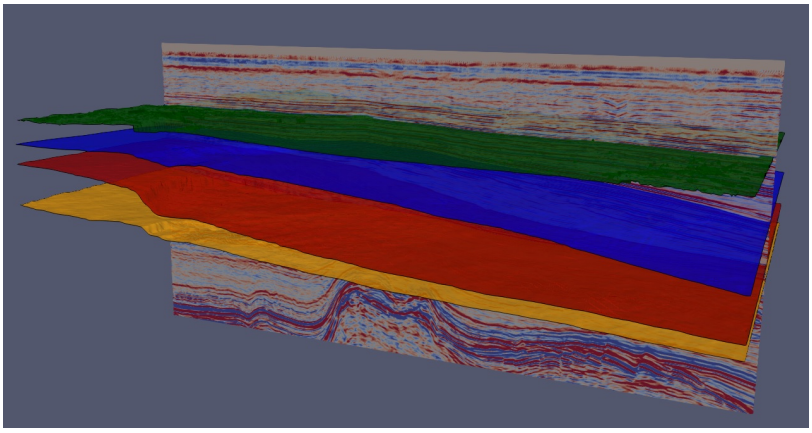

Figure 1.3: Four interpreted seismic horizons shown (horizontal surfaces). From top to bottom, green, blue, red, orange

Medical images capture volumetric information using methods such as magnetic resonance imaging (MRI), computed tomography (CT) and ultrasound imaging. The purpose of medical images is to study the human anatomical structures from volumetric images. A lot of effort in medical imaging is devoted to detect and segment many structures in an automated manner. Examples of a free-boundary surface in medical images includes lung fissures [65, 66, 67, 68, 69], knee cartilage [70, 71, 72], and heart ventricles [73, 74, 75]. Lung fissures are thin structure that separate different lobes in the lung and help in their expansion. Figure 3.16 shows lung fissures. Each lung (left and right lung) has an oblique fissure to separate upper lobe from lower lobe. The right lung has a horizontal fissure to separate upper lobe from middle lobe. Knee cartilage is a coat of tissue that exists where two bones meet as a joint. The detection of knee cartilage can be modeled as an open surface with free boundary. Figure 1.4 right, shows knee cartilage in orange color. The boundary of the surface is shown in red.

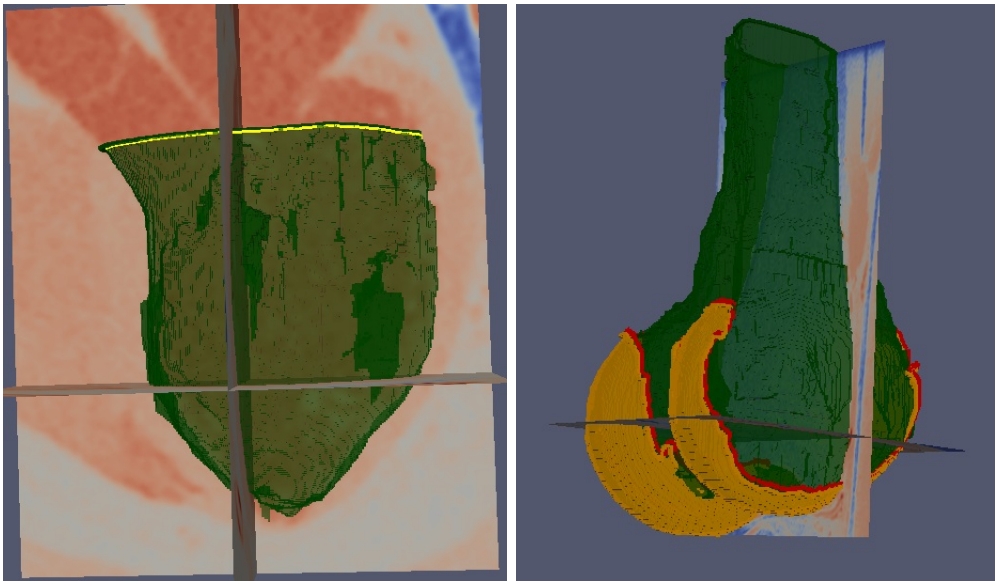

Figure 1.4: Other example application [Left]: Left ventricle surface extraction from CT scan. [Right]: Knee cartilage surface extraction from MRI

Heart ventricles are a cavity or chamber the can be filled with fluid in the human heart. It exists in the lower part of the heart. There are two ventricles in the human heart. The right and the left ventricle. Each ventricle can be modeled as open chamber as can be seen from Figure 1.4, which shows the left ventricle in green color and its boundary in yellow.

Chapter 2

Mathematical Preliminaries

2.1 Introduction and Motivation

In this chapter, the necessary background in topology, computational topology, and Morse theory, will be introduced. These subjects form the basis for my surface extraction algorithm. Roughly speaking, topology is the study of properties of shapes, called *topological spaces*, that are *invariant* to continuous deformations. For example, consider the surface of a donut; a topological property of this shape is that it has one hole. No matter how the donut is deformed (so long as there are no tears), there will always be one hole. Existence of holes is one of many topological properties. My algorithms for surface extraction will extract certain topological structures from data, and thus I will present the necessary background from topology. Topology was initially designed without much thought on how one could infer topological structures from data stored on computers. Computational topology is an area that seeks to construct algorithms for extracting topological structures from *discrete* data. Since the aim is to create practical algorithms, I build on computational topology, specifically, *cubical complexes*. Finally, since the methods will extract topological structures of *functions* defined on manifolds, i.e., certain topological spaces, rather than directly from topological spaces, I review the necessary background material from *Morse theory*. Morse theory studies properties of topological spaces through invariants of certain functions, called Morse functions, defined on the topological space. The algorithms will extract certain topological structures of functions defined on manifolds, through constructions

in Morse theory called the *Morse complex* and the *Morse-Smale Complex*.

2.2 Basic Topology

In this chapter, some of the basic topology definitions that are used to define a deformation retraction are reviewed. A *deformation retraction* is a continuous transformation from one topological space to another that preserves some important topological properties. It is a fundamental operation in my algorithms for surface extraction. The definitions in this section will build up to the definition of deformation retraction.

2.2.1 Topological Spaces and Continuous Functions

I begin with some of the standard definitions that will be used throughout this dissertation. These definitions will help us to define the meaning of topologically equivalent (such as a two-dimensional sphere and a surface of a cube) or topologically distinct (such as a two-dimensional sphere and the surface of a doughnut, i.e., a torus). The definitions from 1 to 4 are largely from [76].

Definition 1 (Topological Space)**.** *A **topological space** consists of a set X and a collection of subsets T of X that satisfy the following properties:*

- $X \in T$, $\emptyset \in T$.
- *(Closed under unions) Given a collection of subsets $U_i \in T$, $i \in J$ in which J is some index set, then $\cup_{i \in J} U_i \in T$.*
- *(Closed under finite intersections) Given a **finite** collection of subsets $U_i \in T$, $i \in J$ where J is some finite index set, then $\cap_{i \in J} U_i \in T$.*

*The elements of T are then called the **open** sets in X, and their complements are called **closed** sets in X. Sometimes a set X with topology T is then referred to as the topological space (X, T).*

Given a topology T for X, the elements of T are said to be **open** sets in X. A subset is called **closed** if its complement in X is open.

As an example, consider the set X consisting of real numbers $\mathbb{R}$. In the standard (or Euclidean) topology on $\mathbb{R}$ we call a subset U of $\mathbb{R}$ open if and only if for every element $p \in U$ we can find an open interval $(a, b) = \{x | a < x < b\}$ so that $p \in (a, b) \subset U$.

The topology on $\mathbb{R}$ can be constructed by declaring that all intervals (a, b) to be open. Then construct all sets obtained from these by forming arbitrary unions of open intervals and finite intersections of these. This then produces all open sets.

As a second example, consider the Euclidean topology on $\mathbb{R}^n$. Here all the n-dimensional open balls, i.e. sets of the form $\{y | ||x_0 - y|| < r\}$ which has center x_0, radius r and where $||.||$ represents the Euclidean distance. Then define a set U to be open if for every $p \in U$ we can find an open ball B so that $p \in B \subset U$.

There is a simple method to also defined a topology on a subset, called the induced or subspace topology which is defined next.

Definition 2 (Induced Topology)**.** *Let S be an arbitrary subset of $\mathbb{R}^n$. The **topology induced** on S consists of the intersection of open sets in $\mathbb{R}^n$ with S. Then a subset V of S is said to be open in this topology if $V = S \cap U$ with U open in $\mathbb{R}^n$.*

As a simple example consider a circle in $\mathbb{R}^2$, centered at the origin with a radius equal to 1. The points on this circle can be parametrized by an angle θ between 0 and 2π. The set of points on the circle for which the angle θ is in the interval $(0, \pi)$ is an open set because it is the intersection of the circle with the set of points (x, y) for which $y > 0$, a set that is open in $\mathbb{R}^2$.

The definition of the topology given above will lead to the definition of continuity that is standard in topology. Since I consider continuous maps on topological spaces in my algorithms, I state the definition here.

Definition 3 (Continuous Maps)**.** *Let X be a set with topology T_X and Y be a set with topology T_Y. A map $h : X \to Y$ is a* ***continuous map*** *if for any open set V in Y (i.e. $V \in T_Y$) its inverse image $h^{-1}(V) = \{x \in X | h(x) \in V\}$ is open in X.*

In my algorithms, we will be deforming sets and it is important that these sets are not broken up into separate pieces. Therefore it is important to 'preserve' the topology, and to be able to say the end-result of the algorithm has the same topology as the set we began with. In topology, these two objects are said to be homeomorphic, which is a concept that is define next. As we will see later this concept is too limiting for our purposes.

Definition 4 (Homeomorphism)**.** *Let X and Y be topological spaces. A* ***homeomorphism*** *between X and Y is a continuous map $h : X \to Y$, which is a bijection and for which the inverse $h^{-1} : Y \to X$ is also continuous. Two spaces X and Y are homeomorphic if there exists a homeomorphism $h : X \to Y$.*

Examples of homeomorphisms are given next [77]:

Example 2.2.1. *A two-dimensional sphere is homeomorphic with the surface of a cube. This is shown by considering a sphere of radius 1 centered at the origin and a cube of side length 2 also centered at the origin. Given a point x on the surface of the cube, we can consider the half-line tx ($t > 0$), which intersects the sphere in a single point $t'x$. One can then show that $h(x) = t'x$ obeys the properties given in the definition.*

Showing that a two-dimensional sphere is not homeomorphic to the surface of a torus is harder, because one needs to show that there is no homeomorphism. A reason why they are not homeomorphic is provided by studying what happens if we delete a single point from the 2-sphere. As defined below, we see that the complement of the point in the two-dimensional sphere can be deformed to a point. The complement of a point on a torus can only be contracted to a union of two circles that meet at a point.

The issue here is that homeomorphic spaces have the same dimension which is restrictive for our purpose becuase my algorithms will be used to deform spaces to lower dimensional spaces.

In fact, topologist uses the concept of a homotopy to define continues deformation. Thus, I define homotopy next [78].

Definition 5 (Homotopy)**.** *Let X and Y be topological spaces. A **homotopy** between X and Y is a continuous map $H : X \times [0, 1] \to Y$. Two continuous maps $f : X \to Y$ and $g : X \to Y$ are homotopic iff there exists a homotopy $H : x \times [0, 1] \to Y$ so that:*

$$H(x, 0) = f(x); H(x, 1) = g(x); \forall x \in X$$

and we write $f \sim g$.

Intuitively this definition means that for a given point $x \in X$, the continuous curve $H(x, t)$ in Y connects $f(x) = H(x, 0)$ to $g(x) = H(x, 1)$.

With this definition at hand, we come to the concept which is also topology preserving in a strict sense, namely a deformation retract [78].

Definition 6 (Strong Deformation Retract)**.** *Let X be a topological space, and A a subset of X then A is a **(strong) deformation retract** of X iff there exists a homotopy $H : X \times [0, 1] \to X$ so that:*

- $H(., 0) = id_X$
- $H(x, 1) \in A \ \ \forall x \in X$
- $H(a, t) = a \ \ \forall a \in A \, \forall t \in [0, 1]$

Thus, a deformation retract is simply a family of continuous maps that continuously "shrinks" or contracts a topological space to a subset of the space.

Example 2.2.2. *The origin* 0 *in* $\mathbb{R}^n$ *is a strong deformation retract of* $\mathbb{R}^n$*. This is easy to see when we define the maps* $H(x,t) = (1-t)x$*. Then* $H(x,0) = x$ *for all* x*, whereas* $H(x,1) = 0$ *for all* x*. If we fix any point* x*, then the curve* $H(x,t)$ *moves the point along its ray to the origin* 0*.*

I now come to the more flexible definition of topological equivalence that is standard in topology, namely that of homotopy equivalence. For any topological space X, let id_X be the identity map on X (mapping a point to the same point) [78].

Definition 7 (Homotopy Equivalence)**.** *Two topological spaces* X *and* Y *are* ***homotopy equivalent*** *if continuous maps* $f : X \to Y$ *and* $g : Y \to X$ *exist so that* $g \circ f \sim id_X$ *and* $f \circ g \sim id_Y$*, where the* $\sim$ *denotes homotopic.*

2.3 Discrete Topology

Since I will be dealing with data (images) that are discrete, my algorithms will be defined in the discrete domain, and so I need to present analogous topological definitions in the discrete domain. Thus, my aim is this section is to mimic some of the previous topological definitions in the discrete domain.

The analog to a topological space in the discrete domain that will form the basis of my computational algorithms is a *cubical complex*. The idea is to create a complex space in discrete domain, we start by linking together simple pieces, analogous to constructing a topological space by simpler sets that induce the topology. In the next section, we will see how these pieces are defined and linked together to form a cubical complex.

2.3.1 Cubical Complexes and the Collapse Operation

My algorithm uses the framework of cubical complexes [79]. This framework allows for performing operations analogous to topological operations in the continuum. It

has been used for thinning surfaces in 3D based on their geometry [80] to obtain skeletons (or medial representations [81]) of geometrical shapes. This is proven to be robust to noise or fine topological features. In contrast to [80], the method is designed to robustly extract ridges of a *function* or data defined on a surface (defined by Fast Marching), rather than geometrical properties of a surface [80] [82] [83].

I now introduce notions from cubical complex theory, which is the basis for the algorithms in future sections. This theory defines topological notions (and computational methods) for discrete data that are analogous to topological notions in the continuum. The notion of *free pairs*, i.e., those parts of the data that can be removed without changing topology of the data, is pertinent to the algorithms. Since the algorithms I define require the extraction of lower dimensional structures (ridge curves from surfaces, and valley surfaces from volumes), it is important that the algorithms are guaranteed to produce lower dimensional structures with correct topology. The theory of cubical complexes (e.g., [84, 80]) guarantees such lower dimensional structures are generated with homotopy equivalence to the original data.

The data (either a curve, surface or volume) will be represented discretely by a cubical complex. A cubical complex consists of basic elements, called *faces*, of d-dimensions, e.g., points (0-faces), edges (1-faces), squares (2-faces) and cubes (3-faces). Formally, a *d-face* is the cartesian product of d intervals of the form $(a, a+1)$ where a is an integer. We can now define a cubical complex (see Figure 2.1) as follows.

Definition 8 (Cubical Complex)**.** *A d-dimensional* **cubical complex** *is a finite set of faces of d-dimensions and lower such that every sub-face of a face in the set is contained in the set.*

The algorithms consist of simplifying cubical complexes by an operation that is analogous to the continuous topological operation of *deformation retraction*, i.e., the operation of continuously shrinking a topological space to a subset, which was presented earlier. For example, a punctured disk can be continuously shrunk to its

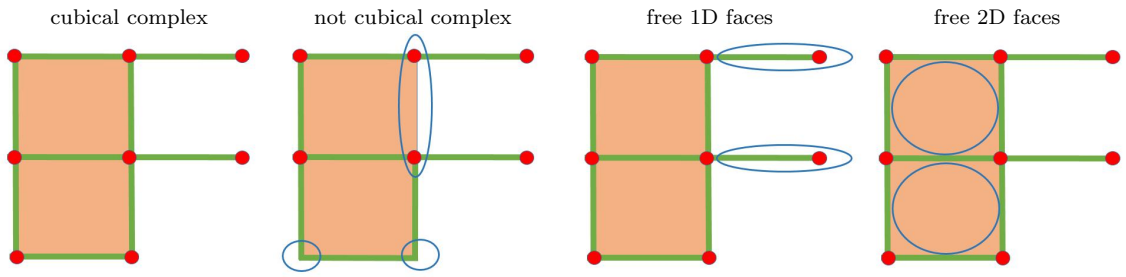

Figure 2.1: [Left two images]: Illustration of faces that form a cubical complex (left) and faces that do not form a cubical complex (0,1,2-faces are marked in red, green and orange). The missing 1-face and 0-faces circled in blue on the right are not in the complex, but they are sub-faces of other faces in the set. [Right two images]: Example of 1-face, 0-face free pairs, and 2-face, 1-face free pairs (circled in blue).

boundary circle. Therefore, the boundary circle is a deformation retraction of the punctured disk, and the two are said to be *homotopy equivalent*. We are interested in an analogous discrete operation, whereby faces of the cubical complex can be removed while preserving homotopy equivalence. *Free faces* (see Figure 2.1), defined in cubical complex theory, can be removed simplifying the cubical complex, while preserving a discrete notion of homotopy equivalence. These are defined formally as:

Definition 9. *Let X be a cubical complex, and let $f, g \subset X$.*

g is a **proper face** *of f if $g \neq f$ and g is a sub-face of f.*

g is **free** *for X, and the pair (g, f) is a* **free pair** *for X if f is the only face of X such that g is a proper face of f. If g is not free, it is called* **isthmus**.

The definition provides a constant-time operation to check whether a face is free. For example, if a cubical complex X is a subset of the 3-dim complex formed from a 3D image grid, a 2-face is known to be free by only checking whether only one 3-face containing the 2-face is contained in X.

In Chapter 3, I construct cubical complexes for the evolving front produced from the Fast Marching algorithm, and retract this front by removing free faces to obtain a lower dimensional ridge curve that lies on the surface that we wish to obtain. I also retract a volume to obtain a valley, which forms the surface of interest.

2.4 Manifolds and Calculus of Functions on Manifolds

The objects of interest in this thesis are surfaces in $\mathbb{R}^3$. In order to extract such surfaces from image data, I will need to use functions defined on surfaces. In this sub-section, I present the definitions for a manifold, which encompasses a surfaces and generalizations to any dimension. Since I will need to extract certain structures that are defined by differentiation of functions on manifolds, I will also introduce differentiable manifolds and basic calculus on manifolds.

2.4.1 Manifolds

Although the intuitive definitions of surfaces involve graphs of functions of two variables, the abstract definition of a manifold is more suitable for our purposes. In this subsection, the definitions and examples from [85]. First I a local set of a coordinate system (also known as a coordinate chart) on a set $S \subset \mathbb{R}^n$ is defined.

Definition 10. *Let S be a subset of $\mathbb{R}^n$ a local coordinate system on S consists of an open set $U \in \mathbb{R}^n$, an open set $W \subset \mathbb{R}^m$ and a 1-1 differentiable function $\phi : W \to \mathbb{R}^n$, $\phi(y_1, ..., y_k) = [\phi_1(y_1, ..., y_m)...\phi_n(y_1, ..., y_m)]^T]$, such that:*

1. *$\phi(W) = S \cap U$*
2. *The derivative matrix $D\phi = [\partial_i \phi_j(y)]$ has rank equal to m at each point $y \in W$.*
3. *$\phi^{-1} : \phi(W) \to W$ is continuous.*

One also says that ϕ gives a local parametrization of the set $U \cap S$.

With this, manifolds that are subsets of $\mathbb{R}^n$ are defined:

Definition 11 (Manifold)**.** *A subset S of $\mathbb{R}^n$ is an m-dimensional closed **manifold** when we have available the following data describing S: A collection of (special) open subsets U_i of $\mathbb{R}^n$ hat cover S: $S \subset \cup_i U_i$. For each U_i a local parametrization $\phi_i : W_i \subset \mathbb{R}^m \to U_i \cap S$.*

Remark. *The manifolds defined in this way are closed in the sense that there are no boundary points. To also allow for boundary points* $\mathbb{R}^m_+ = \{x \in \mathbb{R}^m | x_1 \geq 0\}$ *is defined as the halfspace consisting of points whose first coordinate is non-negative, and the hyperplane* $\mathbb{R}^m_0 = \{x \in \mathbb{R}^m | x_1 = 0\}$. *We then allow for some* i $\phi_i : W_i \subset \mathbb{R}^m_+ \to U_i \cap S$ *to be homeomorphism. The points that are for some* i *in the image of* $\phi_i(W_i \cap \mathbb{R}^m_0)$ *then constitute the boundary of* S.

Remark. *When the dimension* $m = 1$, *we call the manifold a curve instead of a one-dimensional manifold. When the dimension* $m = 2$, *we call the manifold a surface.*

Example 2.4.1. *The graph of a smooth function* $f(x, y)$ *defined on* $\mathbb{R}^2$ *is then a smooth 2-dimensional manifold, via* $\phi(x, y) = (x, y, f(x, y)$. *Similarly, a subset* S *of* $\mathbb{R}^3$ *that is near every point represented as the graph of a function of two variables, which we can represent as* $z = f(x, y)$ *or* $x = g(z, y)$, *etc. is a smooth 2 dimensional manifold in* $\mathbb{R}^3$.

If we suppose that S is a smooth m-dimensional manifold in $\mathbb{R}^n$, described by the coordinate systems (U_i, ϕ_i). we can define a smooth function on S:

Definition 12 (Smooth Function)**.** *A function* $h : S \to \mathbb{R}$ *is smooth if for each* i *the composition* $h \circ \phi_i : W_i \to \mathbb{R}$ *is smooth in the sense of multi-variable calculus, meaning that partial derivatives of all orders of this this function exist.*

Example 2.4.2. *This is an example of a smooth function on a manifold that we will use extensively in this work. Let* S *be a surface in* $\mathbb{R}^3$, *and* P *be a given point in* $\mathbb{R}^3$, *for example, the origin. Define* $h(x) = ||x - P||^2$ *the square of the distance of* x *to* P. *One can verify that* h *is a smooth function on* S. *If the point* P *is not contained in* S, *the function* $h(x) = ||x - P||$ *is also a smooth function on* S,

2.4.2 Calculus on Manifolds

With the previous definition, we are now ready to define derivatives of smooth functions on a manifold. In particular, we can define the differential dh, the gradient ∇h, and the second derivative, the Hessian of h, Hh, of a function $h : S \to \mathbb{R}$, from the surface to the real line. Also, higher order derivatives can be defined, but they are not important for my algorithms. The definitions in this subsection are largely from [86].

To define the first and second order derivatives mentioned in the previous paragraph, the meaning of smooth parametrized curve on the surface needs to be defined. This is simply a mapping $\gamma : (-1, 1) \to S$ from an interval $(-1, 1)$ on the real line to the surface S, so that the intersection of γ with each local coordinate system (U_i, ϕ_i) has the property that $\phi_i^{-1} \circ \gamma$, wherever defined, is smooth.

At any parameter t, the derivative $\gamma'(t) = \frac{d}{dt}\gamma(t)$ then defines a tangent vector to S at the point $\gamma(t)$. When we consider a point $x \in S$ and consider smooth curves γ passing through x at $t = 0$, $\gamma(0) = x$, then the set of all such $\gamma'(0)$ constitutes the tangent space to S at x, and is denoted by T_xS.

Let h be a smooth function on a manifold S for which we need to define the gradient and its Hessian.

Definition 13 (Gradient of a Smooth Function)**.** *Let h be a smooth function on a surface S. We define the* ***gradient*** *$\nabla h(x)$ of the function h at the point $x \in S$ as the tangent vector to S at x which has the following property. Let $v \in T_xS$ be any tangent vector. Let γ be a smooth curve through x with velocity v, i.e., $\gamma(0) = x$ and $\gamma'(0) = v$, then, $\nabla h(x)$ has the property that:*

$$\nabla h(x) \cdot v = (h \circ \gamma)'(0) = \frac{d}{dt} h(\gamma(t)) \text{ at } t = 0.$$

This definition uniquely defines $\nabla h(x)$ as a tangent vector to S at x and is the

direction of maximum increase of the function. These properties agree with those of the ordinary gradient of a function defined in $\mathbb{R}^n$.

With the definition of the gradient of a function h defined on a surface S we can now define the *gradient vector field* of h as solutions of a certain differential equation that is defined on S.

Definition 14 (Gradient Vector Field and Gradient Flow)**.** *Given a smooth function h on a manifold M the* ***gradient vector field*** *is the mapping $\nabla h : M \to TM$, i.e., the mapping that takes a point on the manifold to the gradient of the function at the point, which is located in the tangent space.*

The **gradient flow** *is the solution curve $\gamma : \mathbb{R} \to M$ on M that satisfies the differential equation*

$$\gamma'(t) = \frac{d}{dt}\gamma(t) = \nabla h(\gamma(t)),$$

with initial condition $\gamma(0)$ given.

The **time-t map** *or* **gradient flow map** *is the transformation of M, which maps every point $x_0 = \gamma(0) \in M$ to its location at time t along the gradient flow, $\gamma(t) \in M$.*

An important property of the gradient flow of h is that it increases the value of h along solutions since

$$\frac{d}{dt}h(\gamma(t) = \nabla h(\gamma(t)).\nabla h(\gamma(t)) \geq 0$$

and is equal to zero precisely when $\nabla h(\gamma(t)) = 0$. This happens when the solution is at a critical point of h. For most initial conditions, following the gradient flow for positive time increases the value of the function h. When the manifold is compact, any solution $\gamma(t)$ tends to a critical point as $t \to \infty$ and usually to a different critical point with a lower value of h as $t \to -\infty$.

Another quantity that is important for the development of Morse theory is to describe the type of each critical point of h through its Hessian Matrix, a square

symmetric matrix consisting of second derivatives. For a smooth function $f(t,s)$ of two variables t and s we may consider its mixed partial derivative at $(0,0)$, namely $\partial_t \partial_s f(t,s)$ at $t=0,\ s=0$. To define the Hessian of the function h on a surface S, consider a parameterized family of curves $\gamma(t,s)$ on S, with $\gamma(0,0)=x$, with γ parameterized as $\gamma:(-1,1)^2 \to S$.

Definition 15 (Hessian)**.** *Let $h : S \to \mathbb{R}$ be a smooth function. Let $\gamma(t,s) : (-1,1)^2 \to S$ be a parametrized family of curves with $\gamma(0,0)=x$. Let $\partial_s\gamma(0,0) = v, \partial_t\gamma(0,0)=w$. Both v and w are tangent vectors to S at x The **Hessian** of h at x is the symmetric matrix $Hh(x)$ with the property that always*

$$Hh(x)v \cdot w = \partial_s \partial_t \, h(\gamma(t,s))|_{t=0,s=0}$$

.

Remark. *In the case of a function $h : \mathbb{R}^2 \to \mathbb{R}^2$, the Hessian reduces to the Jacobian of the gradient of ∇h, where the gradient of h on $\mathbb{R}^2$ is just the vector of partial derivatives, as stated below:*

Definition 16 (Hessian in $\mathbb{R}^2$)**.** *The **Hessian** of $h : \mathbb{R}^2 \to \mathbb{R}$ at a point p is defined as the symmetric 2×2 matrix*

$$H_h(p) = \left(\partial_{x_i} \partial_{x_j} h(p) \right)$$

where $i,j = 1,2$ and ∂_{x_i} denotes the partial with respect to the i-th argument of h.

2.4.3 Critical Structures

In this sub-section, I will define certain structures of functions defined on a manifold, which is important for my algorithms in subsequent sections, as the algorithms will extract such structures robustly in order to extract surfaces. In fact, the algorithms

will extract one-dimensional "critical curves" from a function on a surface and a "two-dimensional critical surface" from a function defined on $\mathbb{R}^3$. Such critical structures are where the function obtains certain generalizations of local maxima and minima. Thus, in this sub-section, I present such generalizations of maxima and minima. Note that although I present definitions based on differential operators, my algorithms will compute them differently based on constructions in Morse theory so as to obtain a robust procedure in the presence of noise in real data. In such real data, direct differentiation does not yield robust algorithms. The differential definitions are more intuitive, and by relating them to constructions in Morse theory, I will explain why I have chosen to use constructions in Morse theory for my algorithms, which may not be intuitive.

First, I define critical points, which are where local minima and maxima necessarily occur:

Definition 17 (Critical Point)**.** *A* ***critical point*** x_0 *of a function* $h : M \to \mathbb{R}$ *on a manifold* M *is a point where the gradient vanishes, i.e.,* $\nabla h(x_0) = 0$.

To determine whether the critical point is a local maxima or local minima or a saddle point, one checks the definiteness of the Hessian matrix on the manifold. Indeed, a negative definite Hessian indicates a maxima, and a positive definite matric indicates a minima. Note that a local minima means that locally the function increases in all directions in the tangent space of the point, and a local maxima means that the function decreases in all directions in the tangent space.

In the design of my algorithms, I will also be interested in so called *generalized maximum* and *generalized minimum*, where the function increases or decreases in subspaces of the tangent space rather than the entire tangent space. Let us now give a formal definition of these critical structures. Let V be an nxk matrix with columns $v_1 \in \mathbb{R}^n$ to $v_k \in \mathbb{R}^n$, which denote a basis for the sub-space of directions in defining generalized extrema. Then one can define a generalized extremum as a point such

that all directional derivatives of the function vanish for directions in the sub-space and the Hessian restricted to the sub-space is either positive or negative definite for a minimum or maximum, respectively. The formal definition [87] is the following:

> **Definition 18** (Generalized Extremum)**.** *Let $h : M \subset \mathbb{R}^n \to \mathbb{R}$ where M is an $n-1$ dimensional manifold.*
>
> - *A **generalized maximum** of type $n-1-k$ at x_0 is such that $V^t \nabla f(x_0) = 0$ and $V^t Hf(x_0) V$ is negative definite, where Hf denotes the Hessian matrix.*
> - *A **generalized minimum** of type $n-1-k$ at x_0 is such that $V^t \nabla f(x_0) = 0$ and $V^t Hf(x_0) V$ is positive definite.*

The condition on the Hessian in the definitions above ensure that the generalized extrema is convex (concave) in the subspace of directions spanned by V for minima (maxima). In the algorithms for surface extraction to be presented later, I will extrema generalized extrema that form curves (1-dimensional) from a surface and a surface (2-dimensional) from $\mathbb{R}^3$. The definition above does not by itself guarantee that, for example, a type 1 extrema in dimension $n-1=2$ forms only a curve. For instance, consider the function $h(x,y) = -(x^2+y^2)$ then each point is a type 1 generalized maximum, i.e., moving in the direction tangent to level sets satifies both conditions above. However, the plane is not a one dimensional structure.

One way of specializing generalized extrema to avoid the case in the example above of a type 1 extrema in two dimensions from being a two-dimensional structure is to add additional conditions related to the function's curvature. This can be done by requiring that the matrix V be composed of eigenvectors of the Hessian. In the case of the type 1 maxima, that ensures that the gradient is zero in the direction of an eignevector and that the second derivative in that same direction has greatest absolute value. This greatly restricts generalized extrema. The case of the generalized maxima

restricted to V being eignevectors of the Hessian is called a *ridge* and a generalized minima is called a *valley*. Ridges and valleys are formally defined by [87] as follows.

Definition 19 (Ridge and Valley)**.** *Let $h : M \subset \mathbb{R}^n \to \mathbb{R}$ where M is an $n-1$ dimensional manifold. Let $\lambda_1 \leq \cdots \leq \lambda_{n-1}$ and $e_1, \ldots, e_{n-1} \in T_x M$, be eigenvalues and eigenvectors of the Hessian $Hh(x)$ at $x \in M$. Let $k < n-1$.*

- *A point $x \in M$ is a $n-1-k$* **dimensional ridge point** *of h if $\lambda_k < 0$ and $\nabla h(x) \cdot e_m = 0$ for $m = 1, \ldots, k$.*
- *A point $x \in M$ is a $n-1-k$* **dimensional valley point** *of h if $\lambda_{n-k} > 0$ and $\nabla h(x) \cdot e_m = 0$ for $m = n-k, \ldots, n-1$.*

Although my algorithms are designed to extract one-dimensional generalized extrema that may not be ridges or valleys, I have provided the definitions of ridges and valleys for completeness. I will ensure that a one-dimensional generalized extrema is obtained by constructions in Morse theory, which I show to satisfy the conditions of generalized extrema, and by construction are already one-dimenisonal.

2.5 Morse Theory

The algorithms extract topological structures from functions defined on the image domain and manifolds embedded in the image. I give formal definitions for these topological structures, called the Morse complexes. Intuitively, Morse theory [86] arose from how the topology of level sets of certain "well-behaved" smooth functions change as the level value is changed. For instance, consider the solution set S of a system of equations. One may fix a coordinate direction, for example x_n and consider the subset for which $x_n = c$, where c is a constant. This produces a set S_c, the slice of S at level c in an $n-1$ dimensional space. If we can understand what this set S_c looks like with c varying, then one can reconstruct S from its slices. If one does this for sphere in three dimensions, one would expect to see as possible shapes for

S_c: the empty set, when c is too high or too low, a point, or a circle, and stitching these together one can reconstruct the original sphere itself. As one increases c one follows the gradient flow of the function x_n, which can be used to see how the space is deformed and reduced to simpler pieces.

In the next sub-sections, I give a formal definition of a Morse function, and then the Morse complex, which is defined through gradient flows.

2.5.1 Morse Functions

Morse functions are smooth functions whose critical points are "non-generate":

Definition 20 (Morse Function)**.** *A smooth function $h : M \to \mathbb{R}$ is a **Morse function** if all its critical points are isolated and non-degenerate.*

I now define non-generate:

Definition 21 (Non-degeneracy)**.** *A critical point of a function h is **non-degenerate** if the Hessian of h is non-singular.*

In other words, a point is non-degenerate if none of the eigenvalues of $H_h(p)$ are equal to zero. Therefore we can change coordinates so that the function near the critical point has a simple form:

$$f(x_1, ..., x_n) = c + \sum \lambda_i x_i^2,$$

with λ_i being the eigenvalues of the Hessian. One can change coordinates even further to have the simpler form:

$$f(x_1, ..., x_n) = c + \sum \pm_i x_i^2.$$

The importance of the requirement of non-degeneracy is that one can easily understand how level sets of f change, by knowledge of simple quadratic functions. This

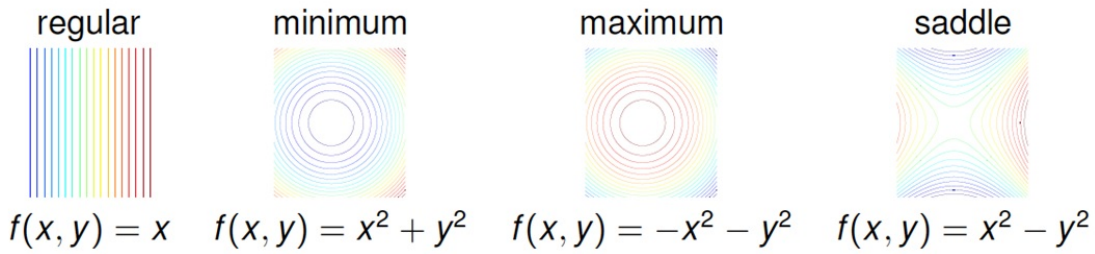

Figure 2.2: Morse Lemma (2D case): level sets in a neighborhood of a point are related by a change of coordinates to one of these forms.

allows for a simple classification of critical points. If a critical point x_0 of f is degenerate its higher order terms in the Taylor series of f at x_0 determine the topological change in shape with changing value of the level. Even including cubic terms the classification is vastly more complicated and thus Morse thoery greatly simplifies the analysis of functions by only considering functions with a non-degneracy condition on its critical points. Indeed, for the two-dimensional case, Morse functions have a simple characterization, as shown in Figure 2.2, which is known in the literature as the *Morse Lemma*. The figure shows the shape of level sets near any point of a Morse function. Note non-degeneracy also implies that critical points are separated from each other since near a critical point the gradient is equal to zero only at the critical point. Therefore there are no curves that consist exclusively of critical points.

Although Morse functions may seem to be overly restrictive, they are in fact good approximations to any smooth function. Indeed, for any smooth function, one can find an arbitrarily close Morse function (in C^1 norm) to the given function. This is because non-degeneracy is not stable: a small perturbation of a dengerate function will make critical points non-degenerate. Such a perturbation may result in more critical points (sometimes fewer) that importantly all have a simple description. This lack of stability also means that one usually does not encounter these situations.

My algorithms in the continuum will be defined for Morse functions, as certain constructions, such as that of the Morse complex will require it. However, when the

algorithms are discretized, one does not need to make any assumptions with respect to the Morse condition in order for the algorithms to function.

2.5.2 Morse Complex

I now define the Morse complex. The Morse complex will allow me to develop a more robust way to extract generalized extrema than computing them with differential operators as suggested by the earlier definitions. Such differential operations are not robust to noise, which is a significant aspect in the applications that I consider.

I will now define the *ascending* and *descending* manifolds of a critical point as all points on a path along the negative (positive, respectively) gradient direction that leads to the given critical point. A path on a manifold M is a mapping $\gamma : [0, \infty) \to M$. A gradient path (defined earlier) is specified by the differential equation $\gamma'(t) = \pm\nabla h(\gamma(t))$, where h is some function defined on M. Formally, the ascending and descending manifolds of a critical point p of h are defined as follows [88]:

Definition 22 (Ascending and Descending Manifolds)**.** *Let $h : M \to \mathbb{R}$ be a function and p be a critical point of h. The* **ascending manifold** *at p is*

$$A(p) = \{x \in M : \text{ there exists } \gamma : [0, \infty) \to M \text{ such that } \gamma(0) = x, \gamma(\infty) = p, \gamma'(t) = -\nabla h(\gamma(t))\}. \quad (2.1)$$

The **descending manifold** *at p is*

$$D(p) = \{x \in M : \text{ there exists } \gamma : [0, \infty) \to M \text{ such that } \gamma(0) = x, \gamma(\infty) = p, \gamma'(t) = \nabla h(\gamma(t))\}. \quad (2.2)$$

For instance, consider the function $h : \mathbb{R}^2 \to \mathbb{R}$ defined by $h(x, y) = x^2 + y^2$. Its

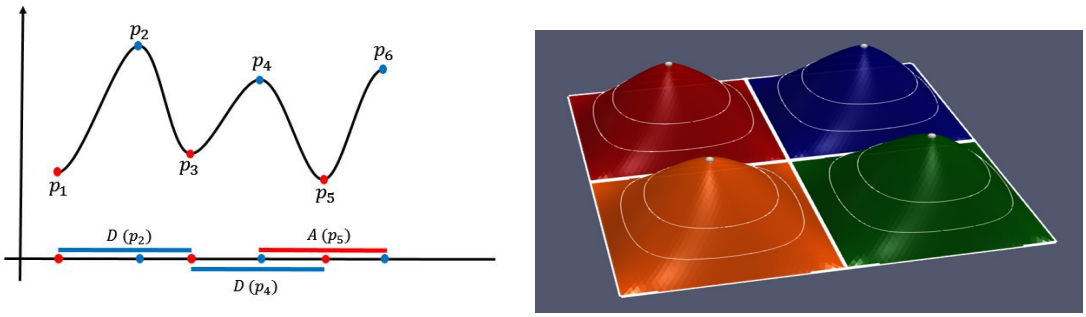

Figure 2.3: [Left]: Ascending and descending manifolds of a one-dimensional function. [Right]: Descending manifolds of a two-dimensional function (each color represents a separate descending manifold).

ascending manifold at the critical point 0 is $A(0) = \mathbb{R}^2$ as all negative gradient paths lead to the origin. Note also that $D(0) = 0$. See Figure 2.3 for visualizations.

The ascending manifolds of local minima decompose the manifold M into disjoint sets. Similarly, the descending manifolds of all local maxima decompose the manifold M into disjoint open sets. The latter decomposition forms the *Morse complex* of h, and the former is the Morse complex of $-h$. I will use the Morse complex in future sections, and thus I state it in a definition below:

Definition 23 (Morse Complex)**.** *Let $h : M \to \mathbb{R}$ be a Morse function. The complex of descending manifolds of f is called the* ***Morse complex****.*

Remark. *I illustrate the meaning of the Morse complex and the cells in the Morse complex when the manifold is a surface.*

- *The 0-dimensional cells correspond to the local minima of h*
- *The 1-dimensional cells correspond to the descending manifolds (curves) of saddle points of h. The ends of these descending manifolds are 'glued' to local minima. The 1-complex then corresponds to the union of local minima and descending manfiolds of saddle points.*
- *The 2-dimensional cells correspond to the descending manifolds (two dimensional surfaces homeomorphic to a disk) of local maxima of h. Their boundaries*

are 'glued' to the 1-complex. We note that usually (but not always) the closure of such a two dimensional a descending manifold is a disk with its boundary containing two local minima m_1, m_2 and two saddle points s_1 and s_2, with the descending manifolds of each s_i end in in m_1 for one branch and m_2 for the other branch. We can then visualize these are quadrangles, see [89, 90].

One can similarly define the ascending Morse complex of h, via its ascending manifolds.

2.6 Summary and Outlook

The purpose of this chapter was to present the background tools that are fundamental for my algorithms for extracting one and two-dimensional topological structures from three-dimensional datasets. Without proper tools from topology, one can easily have much difficulty in designing algorithms. For example, if one desires to extract a one-dimensional structure from a two-dimensional surface, or a two-dimensional surface from volumetric data, one would like to guarantee that the algorithm produces the correct dimension. The framework of cubical complexes and computational topology allows one to have such guarantees. By adopting the computational topology framework and Morse theory, several geometric and topological properties of the extracted objects can be guaranteed even with a high level of noise in the data.

In the next chapter, I will apply the tools from Morse theory and computational topology to design algorithms to extract a boundary curve of a surface and the surface itself from volumetric data. I will show how the use of Morse theory and computational topology makes the algorithms both efficient and robust.

Chapter 3

Surface Extraction: Theory and Algorithms

In this chapter, I will introduce my algorithms for *free-boundary* surface extraction. Recall that a free-boundary surface in this thesis will mean a surface topologically equivalent to a sheet of paper in three dimensions, whose boundary is unknown and must be determined by the algorithms. I build algorithms for free-boundary surface extraction by building on the theory of minimal paths, computational topology, and Morse theory. First, an overview of the methods will be presented. After that, I will illustrate in detail the design of discrete algorithms for extraction of the boundary curve of the free boundary surface. Then, the discrete algorithm for the extraction of the surface will be presented. Although the algorithms were initially designed for free-boundary surface extraction, I will also show how the methods can be adapted to extracting other surface topologies, including closed surfaces (topologically equivalent to a sphere) and surfaces topologically equivalent to a cylinder. This shows the generality of my methodology. To my knowledge, no other framework allows for consistent methodology in extracting all of these different surface topologies.

As one will notice by the end of this chapter, my novel algorithms building on the aforementioned theories are designed to achieve a practical and efficient framework to extract free-boundary surfaces and other surfaces.

3.1 Overview of Method

The input to my algorithm is a single seed point anywhere on the free boundary surface. This is a great advance compared to existing techniques that require an entire boundary curve of the surface as input. The algorithm consists of the following steps (see Figure 3.1):

i) Weighted Distance to Seed Point Computation: From a given seed point on the surface, the Fast Marching algorithm is used to propagate a front to compute shortest path distance from any point in the image to the seed point (Section 3.2.1).

ii) Maximal Curve Extraction: At samples of the propagating front, the maximal curve (a generalized 1-D maxima) of the Euclidean path distance of minimal paths to the seed point are computed by removing points on the front from least to greatest distance while preserving topology (Section 3.2.2). This results in a closed curve that lies on the surface of interest.

iii) Surface Boundary Detection: At snapshots, a graph is formulated from curves from the previous step, and is cut along locations where the Euclidean distance between points on adjacent curves are small, resulting in the outer boundary of the surface when a cost threshold is exceeded (Section 3.2.4).

iv) Surface Extraction: Finally, points in the image excluding the cut curve are removed from highest to lowest based on weighted distance to the seed point while preserving topology - resulting in the desired surface (Section 3.3).

3.2 Surface Boundary Extraction

In this section, I present the algorithm for extracting the boundary curve of a free-boundary surface from a possibly noisy local likelihood map of the surface defined in a

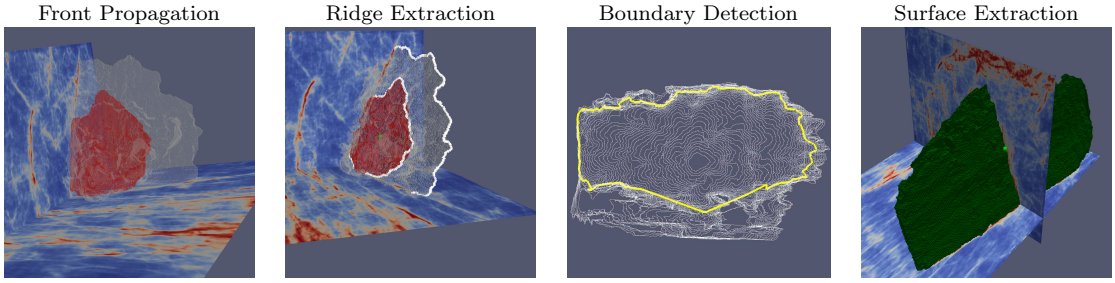

Figure 3.1: Overview of SurfCut. Starting from a user specified seed point on the surface, a front is propagated (left), curves are extracted (middle left), a cut of these curves is performed forming the boundary (middle right), and the surface is extracted (right).

3D image. The algorithm consists of retracting the fronts (closed surfaces) generated by the Fast Marching algorithm to obtain ridge curves on the surface of interest. I therefore briefly review Fast Marching in the first sub-section before defining the novel algorithms for surface extraction. The reader may consult the first chapter for a more detailed exposition of Fast Marching.

3.2.1 Fronts Localized to the Surface With Fast Marching

Fast Marching Method [3] is used to generate a collection of fronts that grow from a seed point and are localized to the surface of interest. I denote by $\phi : \Omega \subset \mathbb{R}^3 \to \mathbb{R}^+$, a possibly noisy function defined on each pixel of the given image grid. It has the property that (in the noiseless situation) a small value of $\phi(x)$ indicates a high likelihood of the pixel x belonging to the surface of interest.

Fast Marching solves, with complexity $O(N \log N)$ where N is the number of pixels, a discrete approximation to $U : \Omega \subset \mathbb{R}^3 \to \mathbb{R}^+$, the solution of the eikonal equation:

$$\begin{cases} |\nabla U(x)| = \phi(x) \quad x \in \Omega \backslash \{p\} \\ U(p) = 0 \end{cases} \tag{3.1}$$

where ∇ denotes the spatial gradient (partials in all coordinate directions), and $p \in \Omega$ denotes an initial seed point. For this situation, p is required to lie somewhere on

the surface of interest. The function U at a pixel x is the weighted minimum path length along any path from x to p, with weight defined by ϕ. U is called the weighted distance. Minimal paths can be recovered from U by following the gradient descent of U from any x to p. A front (a closed surface, which I hereafter refer to as a front to avoid confusion with the free-boundary surface) evolving from the seed point at each time instant is equidistant (in terms of U) to the seed point and is iteratively approximated by Fast Marching. As noted by [2], a positive constant added to the right-hand side of (3.1) may be used to induce smoothness of paths. The front, evolving in time, moves in the outward normal direction with a speed proportional to $1/\phi(x)$. Fronts can be alternatively obtained by thresholding U at the end of Fast Marching. The solution of (3.1) is continuous, and can be approximated as smooth since the solution is a viscosity solution [91], and so a limit of smooth functions.

3.2.2 Contours on the Surface from Front 1D Maxima

If one choose the seed point p to be on the free-boundary surface of interest, the front generated by Fast Marching will travel the fastest when ϕ is small (i.e., along the surface) and travel slower away from the surface, and thus the front is elongated along the surface at each time instant (see Figure 3.3). The algorithm is based on the following observation: points along the front at a time instant that have traveled the furthest (with respect to Euclidean path length), i.e., traveled the longest time, compared to nearby points, lie on the surface of interest. This is because points traveling along locations where ϕ is low (on surface) travel the fastest, tracing out paths that have large arc-length.

This property can be more easily seen in the 2D case (see Figure 3.3): suppose that we wish to extract a curve rather than a surface from a seed point, using Fast Marching to propagate a front. At each time, the points on the front that travel the furthest with respect to Euclidean path length lie on the 2D curve of interest. This

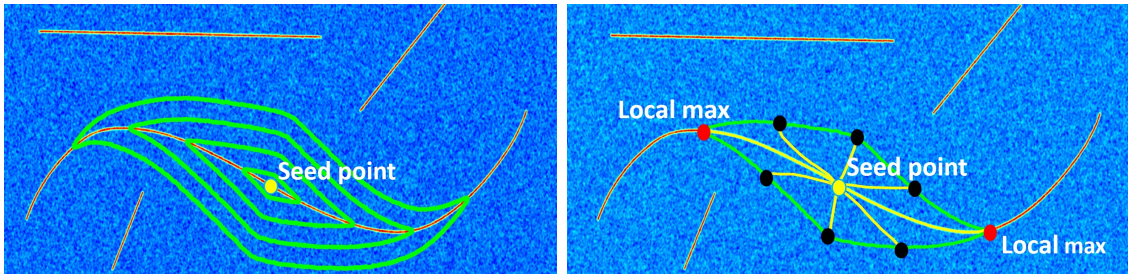

Figure 3.2: Fast march travel in 2D [Left]: Fast marching fronts (green). [Right]: Minimal paths from a front (yellow). It shows that the local maxima of Euclidean length of minimal paths on front lies on curve of interest

has been noted in 2D by [5]. In 3D (see Figure 3.3), we note this generalizes to a *one-dimensional generalized maxima* of Euclidean minimal path length U_E (defined next) are on the surface of interest. The Euclidean minimal path length U_E is defined as follows. Define a front $F = \partial\{x \in \Omega : U(x) \leq D\}$ where ∂ denotes the boundary operator. The function $U_E : F \to \mathbb{R}^+$ is such that $U_E(x)$ is the Euclidean path length of the minimal weighted path (w.r.t to the distance U) from x to p.

Computationally, U_E is easy to obtain by keeping track of another function $U_E : \Omega \to \mathbb{R}^+$ in Fast Marching for U. One follows the ordered traversal of points according to Fast Marching in solving for U, and simultaneously updates the value of U_E based on a discretization of (3.1) with ϕ chosen equal to 1. This gives the Euclidean length of minimal paths determined from U.

The fact that one-dimensional generalized extrema lie on the surface is visualized in the right of Figure 3.3. Points on the intersection of the surface and the front are such that in the direction orthogonal to the surface, the minimal paths have Euclidean lengths that decrease. This is because ϕ becomes large in this direction, thus minimal paths travel slower in this region, so they have lower Euclidean path length. Along the surface, at the points of intersection of the surface and front, the path length may increase or decrease, depending on the uniformity of ϕ on the surface. This implies that points on the intersection of the front and surface are ridge points of $U_E|F$.

I now give an analytic argument that common points to the surface and the front

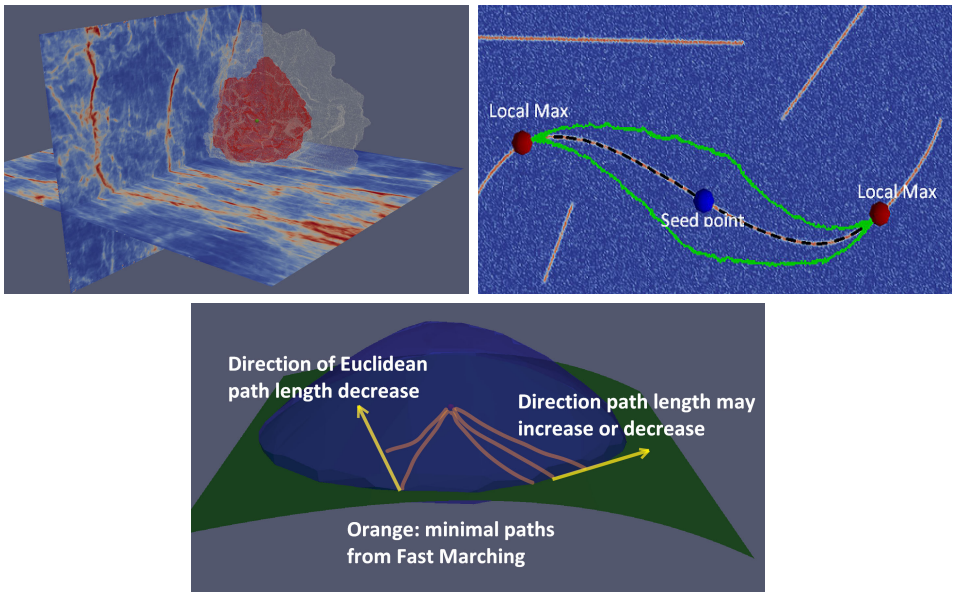

Figure 3.3: [Top left]: The evolving Fast Marching (FM) front at two different time instances in orange and white. The function $1/\phi$ evaluated at x is the likelihood of surface passing through x, and is visualized (red - high values, and blue - low values). The fronts are localized near the surface of interest. Ridge points of U_E, the Euclidean path length of minimal weighted paths, lie on the surface of interest. [Top right]: This is more easily seen in 2D where the local maxima of the Euclidean path length (red balls) of minimal paths (dashed) are seen to lie on the curve of interest. The green contour is a snapshot of the front. [Bottom]: Schematic in 3D with a front (blue), surface (green), and several minimal paths (orange). Orthogonal to the surface where the surface intersects the front, the Euclidean path length decreases. Along the surface, the path lengths may increase or decrease. This indicates a one-dimensional generalized maxima.

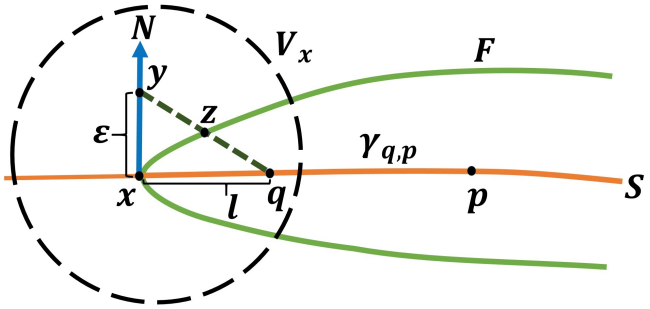

Figure 3.4: Schematic of quantities in the proof of Proposition 1.

are one-dimensional generalized maxima.

Proposition 1. *Suppose $S \subset \Omega$ is a smooth surface and $p \in S$. Consider the front $F = \{U = D\}$ and suppose $x \in S \cap F$ then x is on a one-dimensional maxima of $U_E : F \to \mathbb{R}$, where $U_E(y)$ is defined as the Euclidean length of the minimal path from y to p. One assume that locally ϕ is larger on S than points not on S.*

Proof. Let $x \in S \cap F$ and let N be a normal vector to S at x. We choose a neighborhood $V_x \subset \Omega$ around x so that S is approximately flat and ϕ is approximated as

$$\phi(x) = \begin{cases} K_1 & x \notin S \cap V_x \\ \\ K_2 & x \in S \cap V_x \end{cases},$$

where $K_1 > K_2 > 0$, which are constants. Let us consider a point $y = x + \varepsilon N$, where $\varepsilon > 0$ is small, and the minimal path from y to p (see Figure 3.4). We note that minimal paths within $V_x \backslash S$ will be straight lines as ϕ is uniform in that region. For $\varepsilon > 0$ small enough, one can find $q \in S$ on the minimal path from x to p so that the minimal path from y to p is the straight line path from y to q appended to the minimal path from q to p. It can be noticed that if we let $\ell = |x - q|$ then

$$U(x) = U(q) + K_2 \ell.$$

Also,

$$U(y) = U(q) + K_1\sqrt{\ell^2 + \varepsilon^2},$$

and any point z on the line between y and q will have

$$U(z) = U(q) + tK_1\sqrt{\ell^2 + \varepsilon^2}$$

where $t \in (0, 1)$. If we search for the point z on the line between q and y on the front F, which has $U(z) = U(x)$, we find that

$$t = \frac{K_2}{K_1}\frac{\ell}{\sqrt{\ell^2 + \varepsilon^2}} < 1.$$

Therefore, the Euclidean length of the minimal path from z to p is

$$U_E(z) = \frac{K_2}{K_1}\ell + \text{len}(\gamma_{q,p})$$

where $\text{len}(\gamma_{q,p})$ is the length of the minimal path from q to p. Notice this has less length than the path from x to p, which is $U_E(x) = \ell + \text{len}(\gamma_{q,p})$. Therefore, $U_E(z) < U_E(x)$. So moving in the direction N along F reduces the Euclidean length of minimal paths. This same argument holds for any z within V_x along the direction $-N$ from x. This implies that $x \in F \cap S$ is a one-dimensional ridge point of U_E. □

This tells us that points of the front that are on the surface must be part of one-dimensional maxima, and so we restrict our attention to what we refer to from now on as *maximal curves* on the front as possible points on the surface.

3.2.3 Maximal Curve Extraction Using the Morse Complex

Since computing generalized extrema directly from Definition 19, using differential operators, is sensitive to noise, scale spaces [92, 93] are often used. However, that

approach, while being more robust to noise, may distort the data, and it is often difficult to obtain a connected curve as the maximal curve. Therefore, I derive a robust method by making use of the Morse complex and cubical complex theory to extract the maximal curve of interest from the data U_E. By my construction, it will be easy to see that the generalized extrema are in fact of the right dimension, that is, one-dimensional and does not in particular degenerate to a 2-dimensional structure, which is undesired. Cubical complex theory guarantees the correct topology of the desired maxima (as a 1-dimensional closed curve) in the discrete domain.

Relation Between Maximal Curves and Morse Complex: In the following proposition, we note that certain one-dimensional generalized maxima of a smooth function can be computed by computing ascending manifolds. We assume that M is a 2-manifold. See Figure 3.5 for a visual explanation of this proposition.

Proposition 2. *Boundaries of ascending manifolds of h are generalized maxima of h.*

Proof. Suppose that $x \in \partial A(p_1)$ then for any neighborhood V_x sufficiently small around x, we have that $\partial A(p_1) \cap V_x$ divides V_x, i.e., $V_x = [V_x \cap A(p_1)] \cup [V_x \cap A(p_2)]$ $(p_1 \neq p_2)$ for the case when V_x intersects two ascending manifolds. Note that $-\nabla h(y) \cdot N_2 > 0$ for $y \in V_x \cap A(p_2)$ where N_2 is the inward normal to $\partial A(p_2)$ when V_x is small enough. If this were not the case, then paths following the negative gradient would intersect the boundary $\partial A(p_2)$, which is not the case since they flow into p_2. By a similar argument, $-\nabla h(y) \cdot N_2 < 0$ for $y \in V_x \cap A(p_1)$. Since the function h is assumed smooth and thus the gradient is continuous, we must have that $\nabla h(x) \cdot N_2 = 0$. Further, the function is decreasing away from x along the directions $\pm N_2$ as points in $V_x \backslash \{x\}$ belong to ascending manifolds. Therefore, the point x is a local maximum in the direction N_2. Ridges satisfy this property. Hence, boundaries of the ascending manifolds are one-dimensional generalized maxima. See Figure 3.5 for a visualization of this argument. □

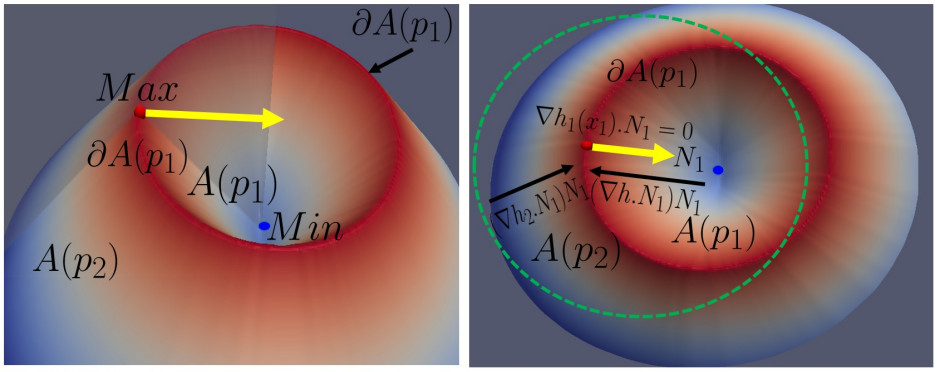

Figure 3.5: Structure of interest and relation to the Morse Complex. Boundaries between ascending manifolds form one dimensional generalized maxima, i.e., maximal curves. [Left]: There are two ascending manifolds $A(p_1)$ and $A(p_2)$ and the boundary between them $\partial A(p_1)$. If ones follows the gradient descent path from any point on the interior of $A(p_1)$ one will go to a unique minima. Also, following the gradient descent on all points on the interior of $A(p_2)$ will lead to a minima on the outside (not shown). [Right]: It can be noticed that the boundary between ascending manifolds meets where the gradient in the direction normal to the ascending manifold boundary is zero. Also, on either side close to the boundary, the gradients in the N_1 direction are increasing towards the maximal curve. Thus, this confirms that such a curve is a one-dimensional generalized maxima. These one-dimensional generalized maxima in the form of curves are what we seek to be on the surface of interest.

Algorithm for Maximal Curves via Morse Complex: Next, I specify a discrete algorithm to determine the Morse complex of $-U_E|F$. The boundaries of ascending manifolds can then be used to extract the relevant maximal curve (1D generalized maxima). The front is retracted to the maximal curve by an ordered removal of free faces based on lowest to highest ordering based on $U_E|F$.

Given a front F, obtained by thresholding the distance U, the two-dimensional cubical complex C_F of the front is constructed as follows. Let $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ be a sampling of a coordinate direction of the image. Then

- C_F contains all 2-faces f in $\mathbb{Z}_n^3$ between any 3-faces g_1, g_2 with the property that one of g_1, g_2 has all its 0-sub-faces with $U < D$ and one does not.
- Each face f of C_F has cost equal to the average of U_E over 0-sub-faces of f.

The algorithm for Morse complex extraction and boundaries of the ascending

manifolds is given in Algorithm 2. The algorithm creates holes at local minima of the function $U_E|F$ defined on 1-faces by removing the adjacent 2-faces. It then removes free faces in increasing order of $U_E|F$ so as to preserve homotopy equivalence. The removed points associated with a local minimum form the ascending manifold for the local minimum. The faces that cannot be removed without breaking homotopy equivalence, i.e., the isthmus faces, form the boundaries of the ascending manifolds. The algorithm removes all 2-faces and preserves only isthmus 1-faces, and hence the remaining structure of C_F is one dimensional. Further, since the algorithm preserves homotopy equivalence, the remaining structure at the end of the algorithm is connected. This is a clear advantage over computation of ridges from differential operators, which does not guarantee connectedness. A heap is used to keep track of the faces in order. The computational complexity of this extraction is, therefore, $O(N \log N)$ where N is the number of pixels, an over-estimate since the faces in the complex are significantly lower than the number of pixels.

Ideally, in the case of clean data ϕ, the *function* U_E defined on the front would have a rather simple topology, indeed a volcano structure (see left image in Fig. 3.6), where the maximal curve separates the inside of the volcano from the outside. The two minimum of U_E on each side of the ridge would correspond to points away from the surface in the direction of the surface normal. In this case, the previous algorithm would produce the inside of the volcano, and the outside as two components of the complex, and the boundary between them as the ridge, as desired. However, due to noise other ridge structures besides the main ridge of interest can be extracted.

Fortunately, the extracted collection of maximal curves can be simplified from the previous algorithm by applying the algorithm iteratively. I construct a new complex with a 2-face for each ascending manifold computed, and a 1-face connecting 2-faces if two corresponding ascending manifolds have intersecting boundaries. Each 1-face in this new complex is assigned a value to be the average of 1-faces in the common

Algorithm 2 Morse Complex Extraction

procedure MORSE COMPLEX(C_F, U_E)
  ▷ *C_F = cubical 2-complex, U_E = cost on 1-faces in C_F*
  id $\leftarrow 0$
  Create heap of 1-faces ordered by U_E (min at top)
  repeat
    Remove 1-face g from heap
    if g is a subset of two faces f_1 and f_2 in C_F **then**
      Remove g, f_1, f_2 from C_F
      $l(f_1) \leftarrow l(f_2) \leftarrow$ id, id $\leftarrow$ id $+ 1$
        ▷ *new id for ascending manifold; hole at local min*
    else if (g, f) is a free pair in C_F **then**
      Remove g, f from C_F
      $l(f) \leftarrow l(f_{adj})$ where $f_{adj} \supset g$ and $f_{adj} \notin C_F$
        ▷ *labels face same as adjacent face containing g*
    else if (f, g) is a free pair in C_F **then**
      Remove g, f from C_F
    else if g is isthmus **then**
      $l(g) = \{l(f_1), l(f_2)\}$ where $f_1, f_2 \supset g$
        ▷ *label is unordered list*
    end if
  until heap is empty
  return C_F, l ▷ *1D maximal curves, labels for 2-faces*
end procedure

boundary between ascending manifolds. The Morse complex of this simplified complex is then computed, and the process is repeated until only one loop remains. The algorithm is given in Algorithm 3. Figure 3.8 shows an example run through this algorithm.

Algorithm 3 Highest Maximal Curve Extraction

procedure HIGHESTMAXIMALCURVE(C_F, U_E)
  ▷ *C_F cubical 2-complex of Fast Marching front*
  ▷ *Euclidean trajectory length U_E defined on 1-faces*
  repeat
    (C'_F, l) = MORSE COMPLEX (C_F, U_E)
    Create 2-cubical complex C''_F with
      a 2-face f for each unique 2-face id in l
      a 1-face g for each unique 1-face id in l
      g joins f_1 and f_2 if $l(g) = \{l(f_1), l(f_2)\}$
    for g each 1-face in C''_F **do**
      $R = \{g' \in C'_F : l(g') = l(g)\}$ ▷ *a maximal curve*
      $U'_E(g) \leftarrow$ average of U_E along R
    end for
    $C_F \leftarrow C''_F$, $U_E \leftarrow U'_E$
  until no degree three 1-faces in C'_F
  return C'_F
end procedure

An example of maximal curves detected for multiple fronts is shown in Figure 3.9. Visual verification that these maximal curves do indeed lie on the surface is shown in Figure 3.10. This procedure of retracting the Fast Marching front to form the main ridge is continued for different fronts of the form $\{U < D\}$ with increasing D. This forms many curves on the surface of interest. In practice, in the experiments, D is chosen in increments of $\Delta D = 20$, until the stopping condition is achieved, and this typically results in $10 - 20$ maximal curves extracted. The next sub-section describes the stopping criteria.

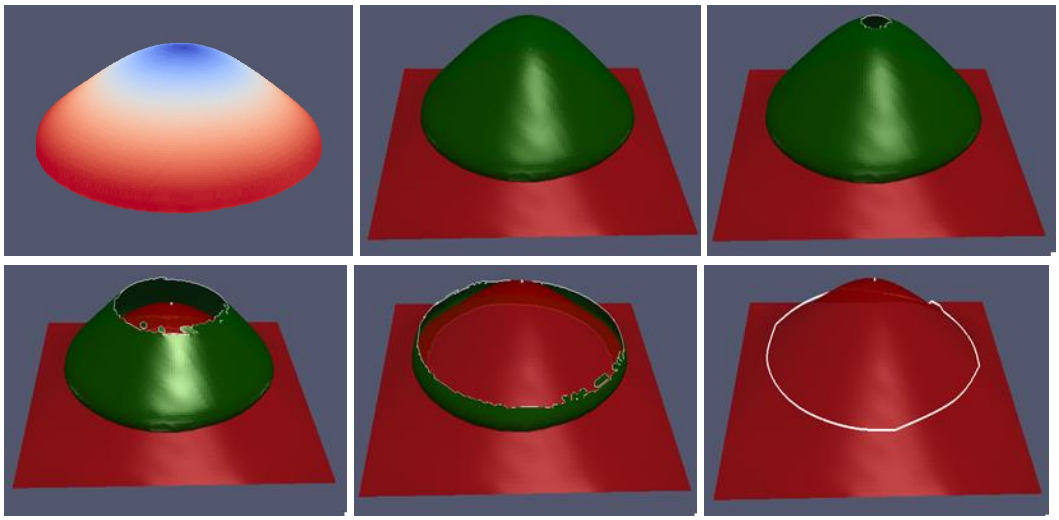

Figure 3.6: [1st image]: Front color coded with Euclidean path length U_E (top view). Red indicates high values. The bottom view (not shown) is a symmetric flip. Topologically, U_E forms a volcano structure (maximal curve, i.e., top of volcano, is darkest red), and inside the volcano is blue. [Subsequent images]: Illustration of iterations (from left to right) of Algorithm 2 on noise-less data to obtain the ridge curve (white) on the Fast Marching front (green) by computing the Morse complex of U_E. The maximal curve lies on the surface of interest (red).

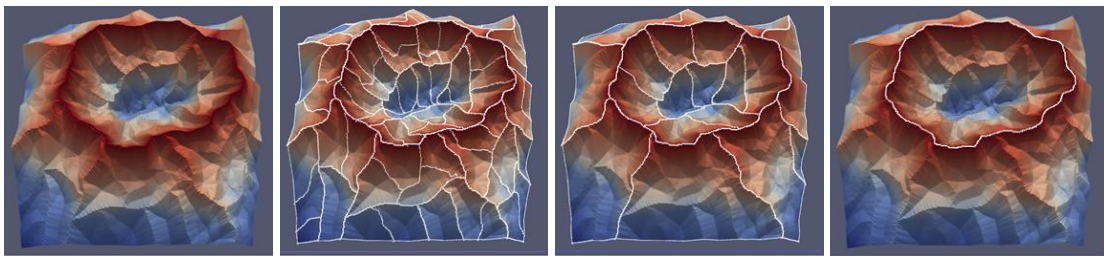

Figure 3.7: Illustration of Algorithm 3 operating on noisy data to obtain the highest ridge.

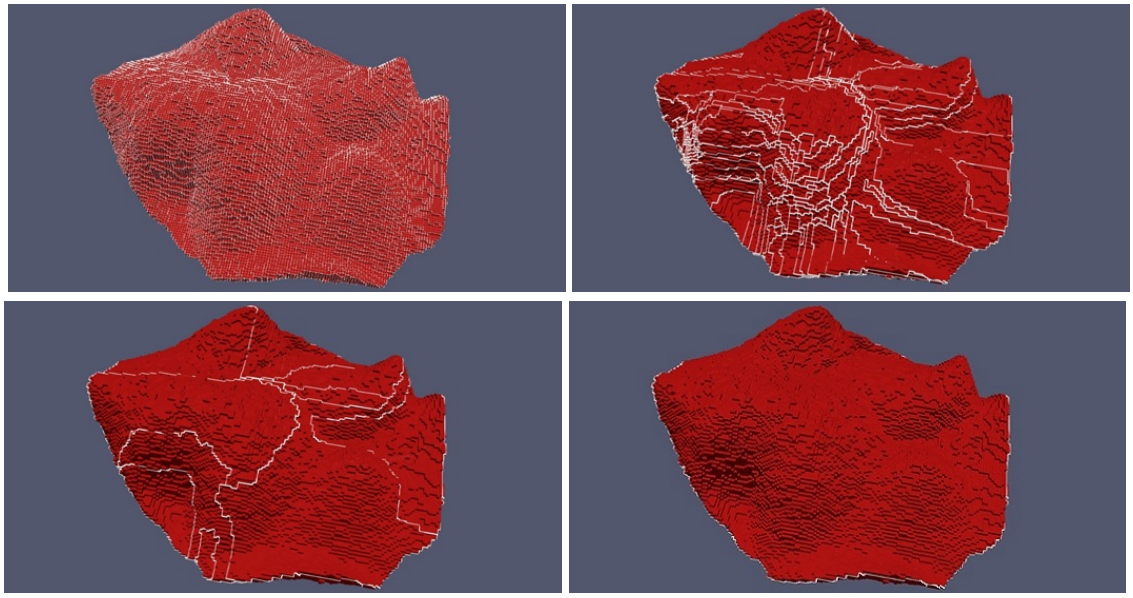

Figure 3.8: Real data example of Algorithm 3 operating on noisy data to obtain the highest maximal curve. In this example, it required three iterations to converge.

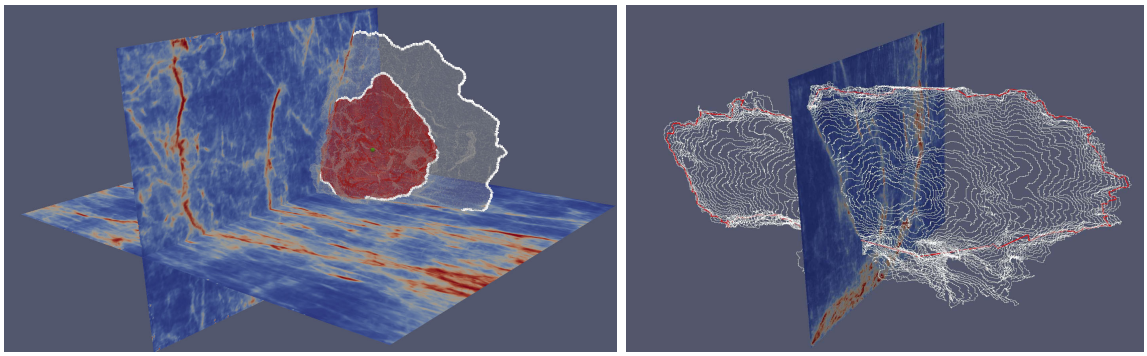

Figure 3.9: [Left]: Maximal curve (white) extraction by retracting the Fast Marching front at two instants. [Right]: An example cut (red) of ridge curves, forming the surface boundary. Notice that the cut matches with the end of high $1/\phi$ (bright areas).

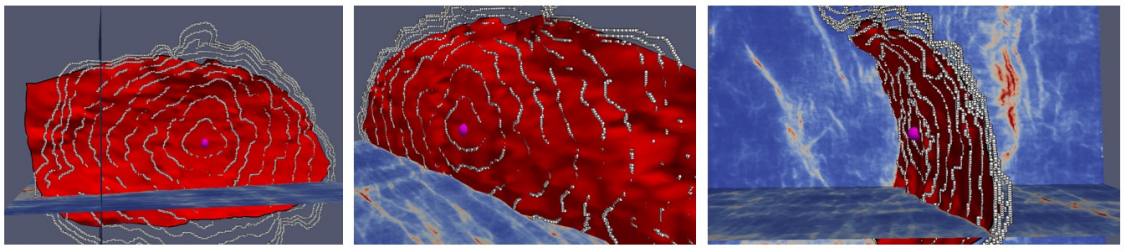

Figure 3.10: Visual verification that the maximal curves are on the surface. The maximal curves are shown along with the surface extracted from the algorithm in the next section, which uses only the boundary curve and not the maximal curves shown.

3.2.4 Stopping Criteria and Surface Boundary Extraction

To determine when to stop the process of extracting maximal curves, and thus obtain the outer boundary of the surface of interest, we make the following observation. Parts of the curves generated from the previous section move slowly, i.e., become close together with respect to Euclidean distance at the boundary of the surface. This is because the speed function $1/\phi$ becomes small outside the surface. Hence, for the curves c_i generated, I aim to detect the locations where the distance between points on adjacent curves becomes small. To construct an algorithm robust to noise, I formulate this as a Graph Cut problem [37].

The graph G is formulated as follows. First, each of the curves extracted is resampled so that all curves have the same number of nodes as the final curve. This operation is done to deal with a shrinking bias of the graph cut algorithm, which I will discuss later.

- vertices V are 0-faces in all the 1-complexes c_i formed from maximal curve extraction after re-sampling
- edges E are (v_1, v_2) where $v_1, v_2 \in V$ are such that v_1, v_2 are connected by a 1-face in some c_i or v_1 is a 0-face in c_i and v_2 is the closest (in terms of Euclidean distance) 0-face in c_{i+1} to v_1
- a cost $|v_j - v_k|$ is assigned to each edge (v_j, v_k) where v_j and v_k belong to different c_i (so that the min cut will be where adjacent curves are close)
- for edges (v_j, v_k) such that v_j and v_k belong to the same c_i, the cost is the minimum Euclidean distance between segment (v_j, v_k) and segments on c_{i+1}
- the source is the seed point p, and the sink is the last maximal curve c_l

The objective is to obtain a cut of G (separating G into two disjoint sets) with minimum total cost defined as the sum of all costs along the cut. In this way, the cut

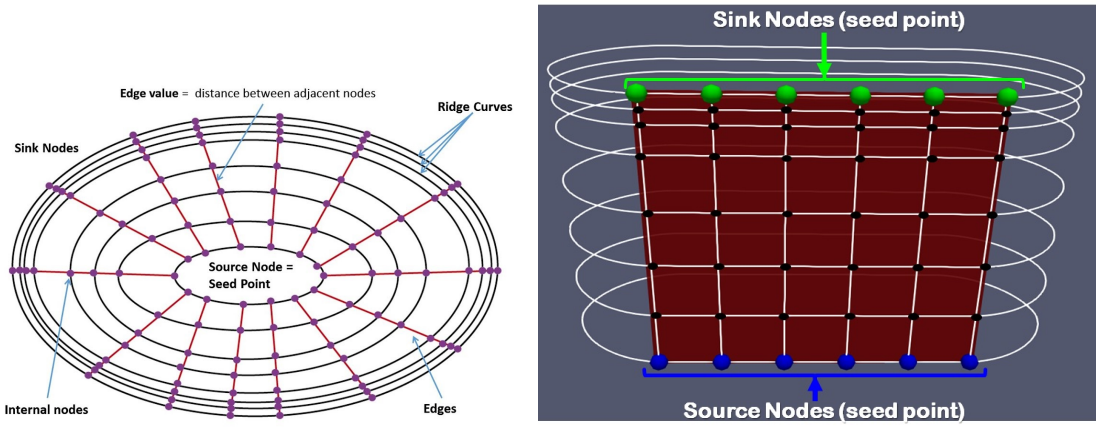

Figure 3.11: Graph formulation for boundary curve extraction from maximal curves. [Left]: Maximal curves that form the graph for the graph cut algorithm. [Right]: Graph in which all maximal curves are resampled to have the same number of nodes.

of the maximal curves along locations where the distance between adjacent maximal curves is small is obtained. The fact that the graph cut optimizes the sum of costs along the cut, explains the resampling of curves mentioned earlier. Indeed, had the resampling not been performed, the graph cut would likely prefer to cut along the curves nearest to the seed point since cuts near the seed point would require only a few edges to separate the graph. Even though those edges have a large distance between adjacent curves, there are only a few edges resulting in a small sum. Thus, the overall sum may be less than cutting along more preferable locations nearer to the final curve where the distance between curves is smaller, but have a larger number of edges. Resampling the curves ensures a similar number of edges near the final and initial curves, avoiding the bias of graph cut to cut only a few edges.

The process of obtaining ridge curves from the Fast Marching front is stopped when the cost divided by the cut size is small; the precise algorithm is explained in the next paragraph. This cut then forms the outer boundary of the surface. As can be noticed from Figure 3.11. The computational cost of the cut (compared to other parts of the algorithm) is negligible as the graph size is typically less than 0.5% of the image. Figure 3.9 shows an example of a cut that is obtained. Figure 3.15 shows

a synthetic example.

The question now is to stop the Fast Marching front evolution in a robust way so that the whole surface is enclosed within the last front, while not traveling too far past the boundary of the surface. Of course, one could just run the algorithm until Fast Marching propagates to the entire image, but this would require much unnecessary computation. To avoid unnecessary computation, as described earlier, one just extracts the maximal curves in a pre-specified thresholded range (time steps) of Fast Marching distance, with large time steps. One still needs to specify the maximum time step. One way to do that is to ask the user for this as an input parameter or put something very high; both cases are not good solutions, as this requires user tuning. To automate the stopping criteria, I used the CUSUM algorithm [94] [95] [96] to detect the right stopping criteria in an optimal manner. CUSUM is a robust algorithm to detect the change mostly used as a method in detection theory. To use CUSUM in the formulation, first, l specify the right criteria for the change that needs to be detected. As we have seen in this chapter, when Fast Marching Front is traveling along the surface, it will go faster than outside the surface. This will lead to maximal curves that are far apart (in terms of ordinary Euclidean distance) when the front is on the surface and maximal curves that are close to each other when it is outside the front.

The previous fact can be used in the CUSUM algorithm to detect when the front reaches the boundary of the surface, and thus terminate the evolution of the Fast Marching front. Specifically, once the cost cut from Graph Cuts changes from high to lower values, the algorithm will terminate. At every time step, Graph Cuts is performed as shown in Figure 3.11.. Let E be the energy of the cut, which is the average value of the distance between adjacent curves for all points on the cut, as specified earlier. E will be computed at every time step so that we will have a vector of values which is the cut energy for each cut. At every time of computation of E we

need to calculate the CUSUM statistic, which is shown below. First, I define average values S_1 and S_2 as:

$$S_{1(k)} = \frac{\sum_{i=1}^{k} E(i)}{k},$$

which is the average of E from 1 to k. Also,

$$S_2(k) = \frac{\sum_{i=k+1}^{N} E(i)}{N-k+1},$$

which is the average of E from $k+1$ to N, which is the current number of maximal curves computed. The CUSUM statistic is now

$$\text{CUSUM}(k) = \max_{k=1,\ldots,N} \sqrt{(S_2(k) - S_1(k))^2}.$$

Notice that the statistic is the difference of averages of cut costs before and after k when the difference is highest in the current data with respect to k. That k that maximizes the difference is called the proposed change point. Once the CUSUM statistic is large enough, indicating a change in the average cut costs, one terminates the extraction of maximal curves and outputs the Graph Cut curve as the boundary of the surface. The CUSUM algorithm is particularly robust due to the fact that it looks not just for changes between adjacent time points, but larger sustained differences determined from all of the data up to the current time. Since the algorithm only deals with scalar values of data points, this algorithm is fast and does not any significant cost to the algorithm, and in fact, saves time since the extraction of extraneous maximal curves is terminated.

3.3 Surface Extraction

I now present the algorithm for surface extraction. Given the surface boundary curve determined from the previous section, I provide an algorithm that determines a sur-

face going through locations of small ϕ and whose boundary is the given curve. The algorithm uses the cubical complex framework and has complexity $O(N \log N)$. The case of surface extraction will involve the determination of two-dimensional generalized minima through the Morse complex.

3.3.1 Surface Extraction Algorithm and Rationale

I show now that the surface of interest lies in a two-dimensional generalized minima of $U : \Omega \to \mathbb{R}^+$, the weighted minimal path length.

Proposition 3. *Suppose $S \subset \mathbb{R}^3$ is a smooth surface and $p \in S$. Let $\phi : \Omega \subset \mathbb{R}^3 \to \mathbb{R}^+$ be a function with low values on S and higher values outside (locally). Then S is a two-dimensional generalized minima of U, where U is the solution of the eikonal equation with $U(p) = 0$.*

Proof. I show that for $x \in S$, U decreases away from x in the direction $\pm N$, the normals to the surface at x. For a small enough neighborhood V_x around x, we may assume that S is flat and that ϕ is approximated by

$$\phi(x) = \begin{cases} K_1 & x \notin S \cap V_x \\ K_2 & x \in S \cap V_x \end{cases},$$

where $K_1 >> K_2 > 0$. We also assume (for now) that x close enough to p so that p lies in V_x. In this case, we see that

$$U(x) \approx U(p) + K_2|x - p| = LK_2,$$

as the minimal path from p to x is approximately a straight line path on the surface, as the surface is nearly flat in V_x. Let $y = x \pm \varepsilon N$ for $\varepsilon > 0$ sufficiently small. We now consider the minimal path from y to p. Note outside the surface, the path must

be nearly a straight line as ϕ is constant. Similarly, on the surface, the minimal path must be a straight line. We see that the minimal path is a straight line between y and some point z on the line joining x to p and then the straight line between z and p (see Figure 3.12). Therefore,

$$U(y) = \min_{\ell} K_2(L - \ell) + K_1\sqrt{\ell^2 + \varepsilon^2}$$

where ℓ is the length of the segment between x and z. The minimizer is $\ell = \varepsilon/\sqrt{1 - r^2}$, where $r = K_2/K_1 < 1$. This yields that

$$\begin{aligned} U(y) &= LK_2 - \frac{\varepsilon}{\sqrt{1 - r^2}}K_2 + \varepsilon\frac{\sqrt{2 - r^2}}{\sqrt{1 - r^2}}K_1 \\ &= LK_2 + \frac{\varepsilon}{\sqrt{1 - r^2}}[K_1\sqrt{2 - r^2} - K_2] > LK_2, \end{aligned}$$

where the last inequality follows from the fact that $\sqrt{2 - r^2} > 1$ and $K_1 > K_2$. Therefore, $U(y) > U(x)$ and so we see a local minimum in the direction N, which implies x lies in a 2-d generalized minima of U. We may now apply the same argument using x to play the role of p, and show that all points in a neighborhood of x on the surface are on a generalized minima. We may continue in this way to show all points on the surface are on the generalized minima. □

Algorithm: One can use the above fact to design an algorithm for extracting the surface (see Figure 3.13). We may perform a deformation retraction of $V_0 = \{U \leq T(0)\}$ where $T(0)$ is chosen to enclose the entire surface, and $T(t)$ is a decreasing function of t. At each time, the points of the level set $L_t = \{U = T(t)\}$ that retain the homotopy equivalence to V_t are removed from V_t. I further impose that the boundary of the surface must not be removed from V_t. This way, all points that are on the surface are retained. One can show this with an inductive argument. Assume for a given time t, the union of all retained sets is a 2-dim set S_{t-} ($t-$ is just before

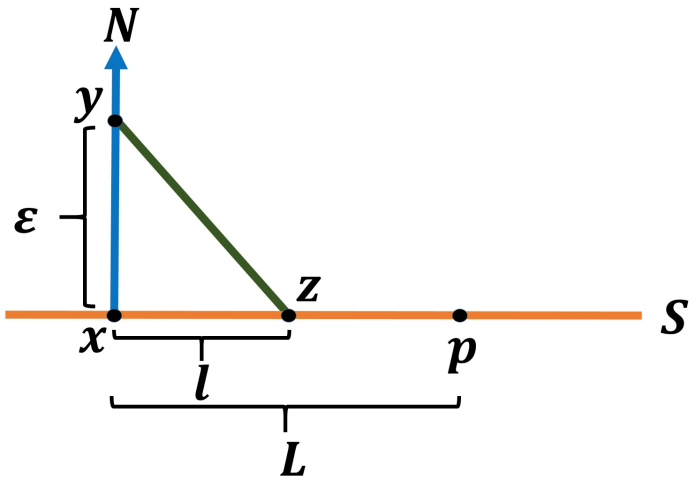

Figure 3.12: Quantities defined in the proof of Proposition 3.

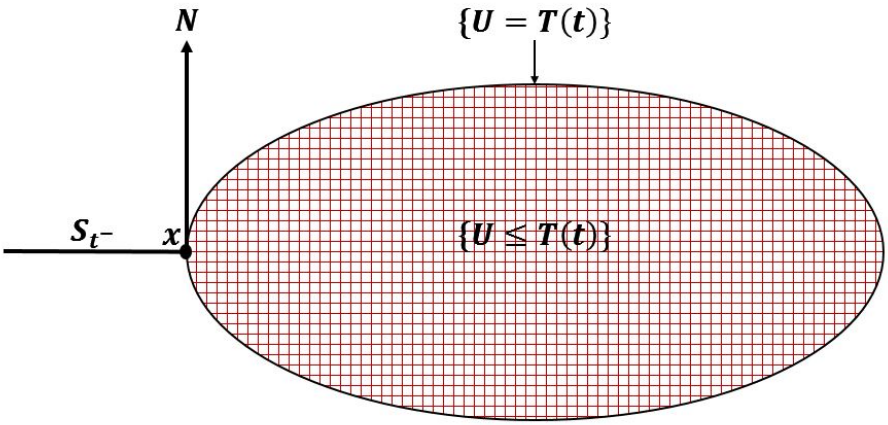

Figure 3.13: Schematic diagram to illustrate surface extraction algorithm defined in the continuum, and the associated quantities in the algorithm definition (see text).

t) that is on the surface, and so $V_{t-} = S_{t-} \cup \{U \leq T(t)\}$. Note that the latter set in the union is a volume. A point $x \in \partial S_{t^-}$ with $U(x) = T(t)$ cannot be removed. Since x is on the surface, which by the proposition is a generalized minima, the normal to the surface at x is tangent to L_t, and U is strictly increasing along the normal. Therefore, removing point x disconnects V_{t-}, not preserving homotopy equivalence. Therefore, $V_t = S_t \cup \{U < T(t)\}$ where S_t contains all points on ∂S_{t^-}.

This procedure can be accomplished with an analogous algorithm in the discrete case. I retract the cubical complex of the image with the constraint that the boundary

curve 1-faces cannot be removed. I accomplish this retraction by an ordered removal of free faces based on weighted path length U. The algorithm is described in Algorithm 4. This results in fronts that have large distance U from the seed point being removed

Algorithm 4 Surface Extraction from Boundary of Surface

```
1: procedure OpenSurfaceExtract(C_I, U, ∂S)
2:                                  ▷ C_I = cubical 3-complex of image, U = FM distance
3:                                  ▷ ∂S = boundary of surface (1-complex)
4:     Create heap of 2-faces ordered by U (max at top)
5:     repeat
6:         Remove 2-face g from heap
7:         if (g, f) is a free pair in C_I for some f then
8:             Remove f and g from C_I
9:         else if (f, g) is a free pair in C_I for some f and g ∩ ∂S = ∅ then
10:            Remove f and g from C_I
11:        end if
12:    until heap is empty
13:    return C_I                   ▷ 2-cubical complex of Valley
14: end procedure
```

first. By the constraint, only the parts of the fronts that do not touch the boundary can be removed. As the removal progresses, faces are removed on either side of the surface. This creates a "wrapping" effect around the surface of interest, which has small values of U. Near the end of the algorithm, points on the surface cannot be removed without creating a hole, so no faces are free, and thus the algorithm stops.

Figure 3.14 shows the evolution from Algorithm 4 to extract the surface from the data used in Figure 3.9. Figure 3.15 shows a synthetic example of the evolution of this algorithm.

3.3.2 Generalized Minima: Surface of Minimal Paths

I now relate the generalized minima that is extracted by the algorithm to minimal paths. I show that the generalized minima, and thus the surface extracted, is a surface formed from a collection of minimal paths to p. First, I show that the gradient path starting from a point in the generalized minima stays in the generalized minima.

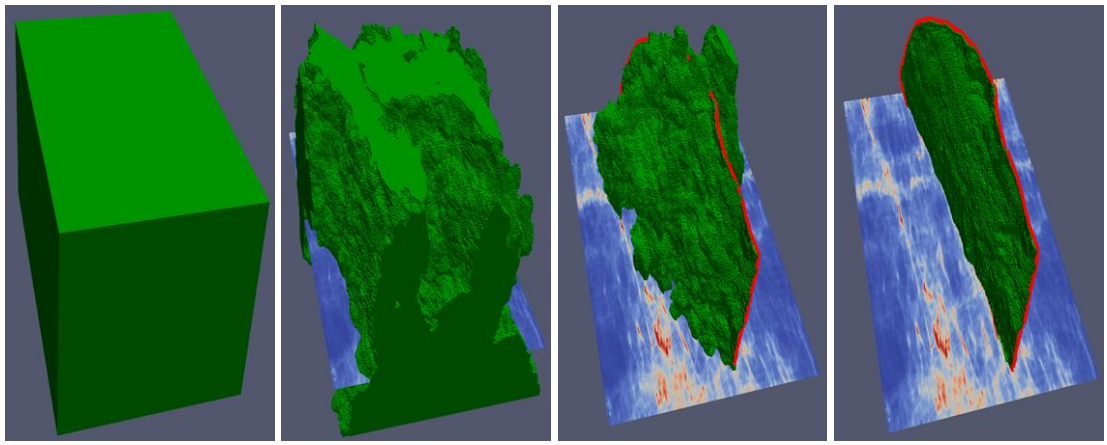

Figure 3.14: Illustration of valley extraction by Algorithm 4, which retracts the volume while preserving 1-faces on the surface boundary (red). This gives the surface of interest.

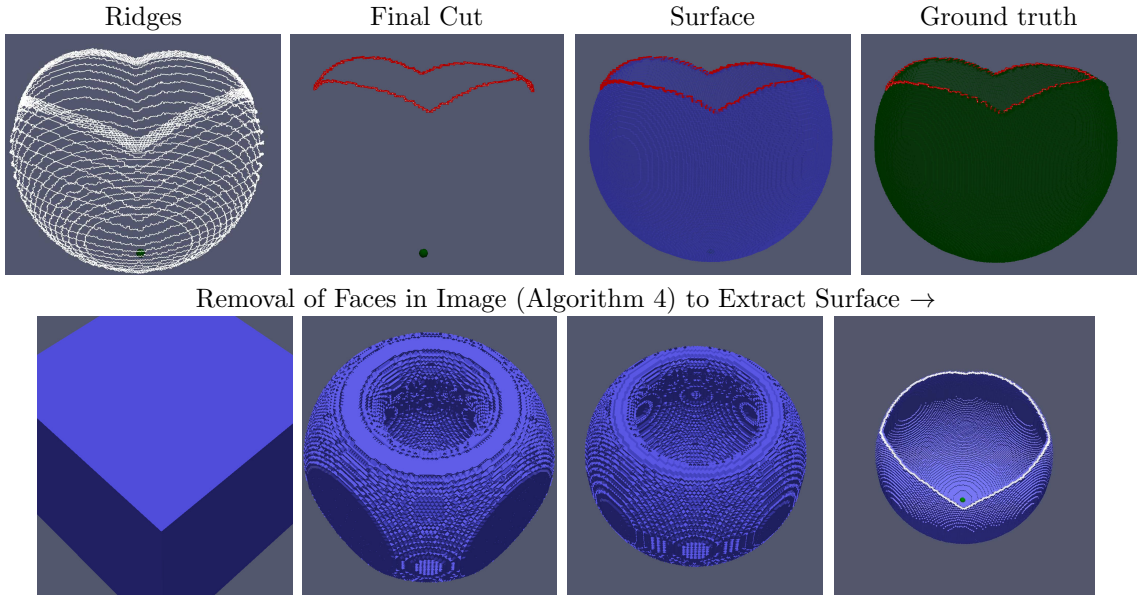

Figure 3.15: Synthetic example of extracting a sphere with top cut such that the boundary is four arcs. The image (not shown) is a noisy image of the cut sphere with holes. Maximal curves are extracted via Algorithm 1 (top left). The final cut of maximal curves (top, middle left), the final surface extracted via Algorithm 2 (top, middle right), and the ground truth (top, right) are shown. Snapshots in the removal of faces in Alg. 2 are shown (bottom), resulting in the surface (right).

Proposition 4. *Suppose $x \in M$ is a generalized minimum point of $h : M \to \mathbb{R}$, then the path γ determined by the gradient descent of h with initial condition x lies on the valley of h containing x.*

Proof. For simplicity, we assume $M = \mathbb{R}^3$ and that the generalized minima is two-dimensional. By definition of a 2D generalized minima in $\mathbb{R}^3$, we have that $\nabla h(x) \cdot N_x = 0$ and $N_x^T Hh(x) \cdot N_x > 0$ for some unit direction $N_x \in \mathbb{R}^3$ where Hh denotes the Hessian. For every neighborhood V_x of x sufficiently small, there exists $y \in V_x$ such that $\nabla h(y) \cdot N_y = 0$ and $N_y^T Hh(y) \cdot N_y > 0$ for some N_y. If that were not the case, then x would be an isolated critical point, which is not the case. By smoothness of h, N is a smooth function. Let S be the points the satisfy the conditions on the gradient and Hessian in V_x.

We consider the path γ defined by the gradient descent of h starting from x. Then by definition of γ and Taylor expansion of h,

$$\nabla h[\gamma(\Delta t)] \approx \nabla h[x - \Delta t \nabla h(x)] \approx \nabla h(x) - \Delta t Hh(x) \cdot \nabla h(x).$$

Taking the dot product of the above with $N_{\gamma(\Delta t)} \approx N_x$, by a Taylor expansion, we have

$$\nabla h[\gamma(\Delta t)] \cdot N_{\gamma(\Delta t)} \approx -\Delta t N_x^T Hh(x) \cdot \nabla h(x).$$

Note that $N_x^T Hh(x) = \lambda N_x^T$ with $\lambda > 0$ since N_x is an eigenvector of $Hh(x)$ if we use a more stringent requirement of a valley. Since $N_x^T \nabla h(x) = 0$ by the definition of generalized minima, we have that $\nabla h[\gamma(\Delta t)] \cdot N_{\gamma(\Delta t)} \approx 0$. Also, $N_{\gamma(\Delta t)}^T Hh[\gamma(\Delta t)] \cdot N_{\gamma(\Delta t)} > 0$ as $\gamma(\Delta t) \in V_x$. Therefore, $\gamma(\Delta t)$ is also in the valley, and thus continuing this way, we can show that the path γ formed from the gradient descent is also in the generalized minimum. □

Using the last property, I show that the surface extracted by the algorithm is a collection of minimal paths to p.

Proposition 5. *Suppose V is a generalized minima of U, the solution of the eikonal equation, containing the seed point p used to define U. Then V is a union of minimal paths to p.*

Proof. Let $x \in V$ then the path γ_x formed from the gradient descent of U starting from x stays in V by Proposition 4. The path γ_x is also a minimal path since gradient paths of U are minimal paths. Note that γ_x ends at p. Therefore, we see that V is the union of γ_x over all x. □

3.4 The Case of Intersecting Surfaces

In real-world scenarios, there could exist cases were multiple surfaces intersect. As an example, Figure 3.16 shows a case in lung CT scan. It can be noticed that the lung has two intersecting surfaces, which are known as fissures. In this case, my algorithm described earlier would extract both surfaces as one surface. I would like to have each surface as a separate surface. To solve this issue, I enforce that the maximal curves have small curvature. This assumption will be realistic in the intersecting scenario. The idea is that when the maximal curves approach the intersection, the curve will change topology from a simple loop to two loops. I will simply discard the loop that causes the curvature of the existing curve to become large. In this way, I will simply get the flattest surface that passes through the intersection. In many of the applications that I am interested in, such as faults in seismic data and lung fissures, this assumption is valid.

It turns out that one can employ an easy method to extract the relevant loop by "projecting" the curve in time step $N-1$ to the next Fast Marching Front (time step N) and add a constraint such that isthmus edges must be in a limited band around the projected curve. In this way, I enforce the small curvature assumption by making sure that the maximal curve in time step N is close to the previous one from time step $N-1$. Thus, the only change to the Algorithm 3 is that I add another

Figure 3.16: Surface intersection in lung data. Right lung contains, one surface. Left lung contains, two intersecting surfaces that require the surface intersection handling.

condition to preserved edges to be in the band and discard edges outside. Previously, the condition to preserve an edge is just to be an isthmus edge. The new update to this is that it is required to be within a specified band around the projected curve in addition to the isthmus condition. Thus, both conditions have to be satisfied in order for edge to go in the preserved edge list. Adding such a constraint is an easy task but one need to be able to project the ridge curve from time step $N-1$ into the new Fast Marching front of time step N efficiently. Thus, I discuss "projection" procedure next.

Given a curve from previous time step and a new Fast Marching Front, our task now is to project the curve into the new Fast Marching Front. The direction of the projection should follow from the basic assumption that the surface has a smooth curvature. Therefore, additional information, which is the direction of projection is required. In fact, for every point on the curve of time step $N-1$, one can use the closest point on the curve at time step $N-2$ to obtain the direction information. This direction information will be used to propagate forward the previous curve, thus obtaining the projection.

3.5 Generalizing Surface Extraction to Other Topologies

The motivation behind developing the methods in previous sections is to solve the challenging problem of extracting free boundary surfaces (open surfaces). However, the method is more general and can handle several topologies. In this section, I show the methods can be easily adapted to handle the cases of closed surfaces and cylindrical topologies. Both of these cases are important for various applications, as described in the introduction. For a proof of concept, I constructed some synthetic images for these topologies to show how one can adapt my existing methodology. I have added noise is added to the 3D images to show that my methods are robust to noise. The next sections will deal with each of the aforementioned topologies.

3.5.1 Closed Surface Extraction

In fact, one can detect the closed surface directly by determining a valley-like structure (2-dim generalized minima) and by-passing the boundary curve extraction algorithm described in the previous sections. Although there are other methods for extracting closed surfaces, such as Graph Cuts, I believe there are advantages to my approach. First, my method has complexity $O(N \log N)$ and this is lower than Graph Cuts, which tend to have a cubic complexity or greater. Second, I have found that Graph Cuts is sensitive to the attribute used and different levels of noise or contrast might change the extracted surface. Preliminary experiments indicate that my method is less sensitive to changes in the attribute.

Since a closed surface does not have a boundary curve, it is not required to use the boundary curve extraction algorithm described in the previous sub-section. From here, I follow similar steps proposed for surface extraction in the previous sub-section. Therefore, a seed point needs to be picked somewhere on the surface as shown in Figure 3.17. After that, the Fast Marching Front is computed from the seed point. It can be observed very easily from Figure 3.17 that the closed surface will form a generalized minima surface (2-dim generalized minima) in the 3D image, which is also the case in free boundary surface extraction. Thus, one can just apply the algorithm used for extraction of the 2-dim generalized minima for the open surface to extract the generalized minima closed surface with no major difference. The algorithm for closed surface extraction is introduced in Algorithm 5. Note that outer loop is required, since similar to the case of maximal curve extraction where multiple curves can arise from the Morse complex, multiple surfaces can be obtained do to noise or other fine structures in the image, and thus simplification must be performed (see Chapter 4). The algorithm effectively just extracts descending manifolds of the 3D data, and simplifies the Morse complex until only one surface remains.

Algorithm 5 Closed Surface Extraction

1: Input: I (3D image), and seed point p on the surface of interest.
2: Compute Fast Marching distance with $U(p) = 0$. See Figure 3.17 in the middle.
3: **repeat**
4: Sort faces from maximum to minimum based on U.
5: **repeat**
6: Remove free and maxima faces from the ordered list. See Figure 3.17 on the right.
7: **until** no more free faces can be removed.
8: Simplify Morse complex
9: **until** one closed surface (one face) remains. See Figure 3.18.

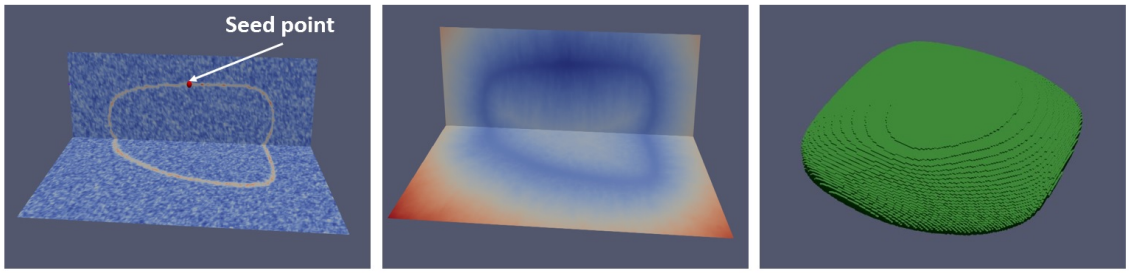

Figure 3.17: Closed surface extraction. [Left]: Input a 3D image containing closed surface and a seed point picked by the user anywhere on the object of interest. [Middle]: Fast Marching algorithm is computed from seed point $U(p) = 0$. [Right]: Extracted closed surface obtained by extracting boundaries of descending manifolds.

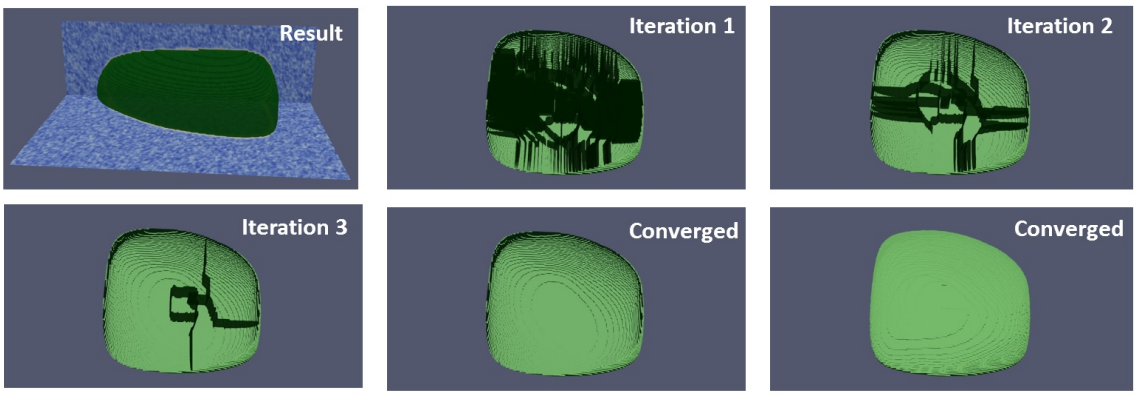

Figure 3.18: Visualization of the iterations of the algorithm for extraction of boundaries of descending manifolds. Since the data are noise, this example required four iterations to converge. Note that the closed surface is cut from the middle for the purpose of visualization to see how the shape is simplified at each iteration of the algorithm.

3.5.2 Cylindrical Topology Surface Extraction

Similar to the free-boundary surface extraction, cylindrical topology surface extraction requires boundary curves. The main difference is that the cylindrical topology requires two boundary curves (arising form its boundary) instead of one in case of the simple open surface. Despite this, this does not make much of a difference in the algorithm. Therefore, they both follow the same steps. Thus, starting with a 3D image and two planar boundary curves picked in two 2D planes as shown in Figure 3.19, I compute Fast Marching from one of the boundary curves. After that, as we have observed in the previous cases, the object will form 2-dim generalized minima in Fast Marching distance. This results in the same algorithm as in the previous cases to be applied on Fast Marching distance. The algorithm for cylindrical topology extraction is introduced in Algorithm 6.

Algorithm 6 Cylindrical topology extraction

1: Input: I (3D image), and the two boundary planar curves of the cylindrical object. See Figure 3.19 on the left.
2: Compute Fast Marching distance with $U(c_1) = 0$ where c_1 is one of the boundary curves.
3: **repeat**
4: Sort faces from maximum to minimum based on U.
5: **repeat**
6: Remove free and maxima faces from the ordered list.
7: **until** no more free faces can be removed.
8: Simplify Morse complex
9: **until** no more than one surface is obtained. See Figure 3.19 middle and right.

3.6 Summary and Outlook

I have demonstrated in this chapter that methods from minimal paths, Morse theory and computational topology can be used to extract surfaces from three-dimensional data. To the best of my knowledge, this is the first time that this has been done. Of particular importance was the Morse complex and the boundaries of ascending and

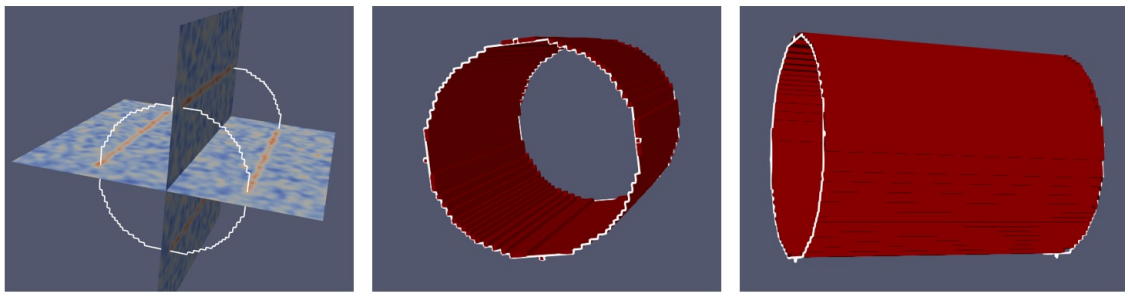

Figure 3.19: Cylindrical topology extraction. [Left]: Input a 3D image containing cylindrical surface and closed curves on each ending slice. [Middle and Right]: Extracted cylindrical topology (extract boundaries of descending manifolds).

descending manifolds that allowed for convenient algorithms to extract generalized extrema. Although I started first with the case of free-boundary surfaces, the last section has shown that the same methodology can also be used to handle other surface topologies. Previously in the literature, cylindrical, closed surface and open surface topologies have all been treated with very disparate mathematical methodologies. It is quite interesting and promising that the framework I have developed here encompasses all these topologies in a constant framework of Morse theory and computational topology.

In the next chapter, I delve into the details of the implementation of the topological operations and the data structures needed to make for an efficient implementation. I include this, since significant thought has been invested so that the resulting algorithms are efficient in terms of both computational cost and memory.

Chapter 4

Implementation Details of Topological Operations

In this chapter, I introduce an efficient implementation of the algorithms, introduced in the previous chapters, to extract the Morse complex and the surfaces from a 3D image. Based on cubical complexes and the deformation retract operation discussed earlier, I introduce the main building blocks, i.e., data structure constructions and processing details, for extracting the Morse complex and related algorithms. These building blocks include cubical complexes construction and processing, implementing the collapse operation and iteratively performing the collapse operation to extract generalized extrema. By introducing these building blocks, I will be able to abstract the commonality of all the operations discussed in Chapter 3 to extract and process both generalized minima and maxima from the Morse complex. Therefore, the same code will be used for extracting both the boundary curve of the surface and the surface. The description will enable the interested reader to efficiently code the algorithms him/herself. In fact, I have investigated several aspects of performance optimization and tried to pick the right level of compromise between speed and memory usage.

4.1 Cubical Complex Storage and Access

To have an efficient implementation of the collapse operation, we need to design the data structure storing the cubical complex of the shape of interest in such a way that it is easy and efficient to access faces adjoining a given face in the complex. To

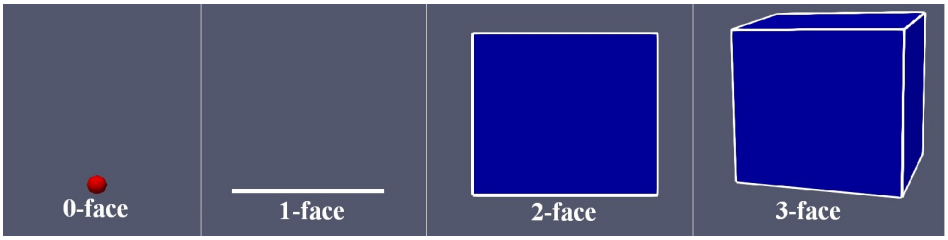

Figure 4.1: Cubical complex elements in 3D.

accomplish this, I use the 3D grid of the image to create indices for points, edges, faces and cubes (see Figure 4.1) and then I create lists for such elements that are present. I pay particular attention to indexing and linking faces of the complex in a flexible way so that accessing and manipulating them can be performed in an efficient manner. Although the implementation is specifically for structures arising in 3D images, it will be obvious that the implementation can easily be adapted to structures embedded in any dimension.

4.1.1 Data Structures

In 3D images, it is known that every voxel is connected to the six neighboring voxels, which I call +X, +Y, +Z, -X, -Y, -Z (see Figure 4.2) based on the canonical coordinate directions. Thus, given this information, we can utilize this for efficient encoding of the n-D faces indices. First, for edges, at every voxel, we need to generate indices for three edges pointing in +X, +Y, +Z. This will then cover each edge in the grid; note the negative directions are not needed as that will lead to having multiple indices for the same edge. Thus, due to the underlying image grid, only a voxel and direction are needed to specify an edge. The price to pay for storing just this small information is that the edges active in a complex will need to be specified with three indicator functions on the entire grid. But, it can be observed easily that this is not much compared to the size of the image, and so we will use this strategy. Faces will be stored in a similar fashion (see Figure 4.2).

Now, I lay down the main data structures used to construct and link the cubical

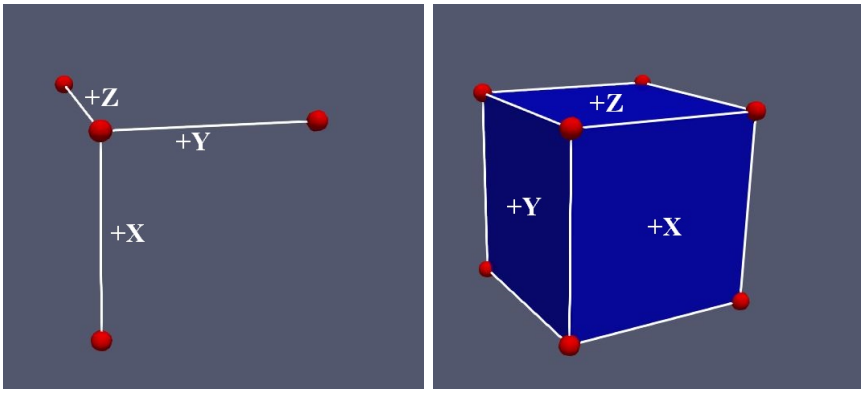

Figure 4.2: Edges and faces index types in 3D images [Left]: Edges index types. [Right]: Faces index types.

complex framework. Mainly, the data about nodes, edges, faces, and cubes are put in 1-D vectors that indicate whether a given cell is active or not. The indices of every vector (array index) are to give the identity to every element. Also, two 2-D arrays are used to accelerate the access to the neighboring edges and faces; I call these arrays `edge_faces_link` and `face_cubes_links`. Below is a list describing each element in the cubical complex data structure:

- `N0` (`size: XSize * YSize * ZSize` , type: 1D boolean array): Original 3D image nodes vector. This vector is used to indicate whether a specific voxel is active or not in the original image. The order of the voxel in the vector is served as the index of the voxel. The length of this vector is the same as the number of voxels in the original 3D image. The datatype size must be at least 1-byte (C/C++ representation of bool data type).

- `N1` (size: `active nodes`, type: 1D integer array): This vector contains the indices of the active voxels only. The purpose of this vector is to limit the cubical complex to be built only on the active voxels and as a result to limit the processing of the cubical complex elements to the active voxels. This is important when I process a subset of voxels of the original image in which the subset is much smaller than the original image. This results in a major speedup

and a better efficiency. In this case, it is important when extracting maximal curves. The data size for the subsequent elements such as edges, faces, and cubes with their linking vectors will depend on the size of this vector. Moreover, the speed of sorting and collapse operation will depend on the length of this vector. In the case in which all voxels in the original 3D image are active, then this vector can be eliminated.

Therefore, the `N0` and `N1` vectors are used to translate between the voxels on the original 3D image to the active voxels. The edges pointing down (+X) are put in first in the edges vector followed by edges pointing to the right (+Y) followed by edges pointing back (+Z). Thus, the vector length for edges and faces is equal to the length of active voxels multiplied by 3. A similar methodology will be used to represent the active faces. With these arrays, I do not store the explicit spatial location for any cell, and I have an effective way to navigate through the cubical complex elements.

- `Type3` (size: `3 * active nodes`, type: 1D integer array): This data structure is used to indicate the edges and faces type, which is one of +X,+Y,+Z. We can notice from Figure 4.2 that there are three directions for the edges and faces, to uniquely represent all edges and faces in the grid. This is used for indexing faces and edges and to accelerate the navigation across different n-faces (edges, nodes, faces, and cubes). An example of the indices used to label the edges, faces and cubes are given in Figure 4.3.

- `E` (size: `3 * active nodes`, type: 1D boolean array): This data structure is the edges vector. It stores primary edges information. It will indicate whether the edge is active or not. The status of all edges at the beginning is active. During the collapse operation, some of the edges are removed as a result of collapse operation. The collapse operation will turn some values of this vector

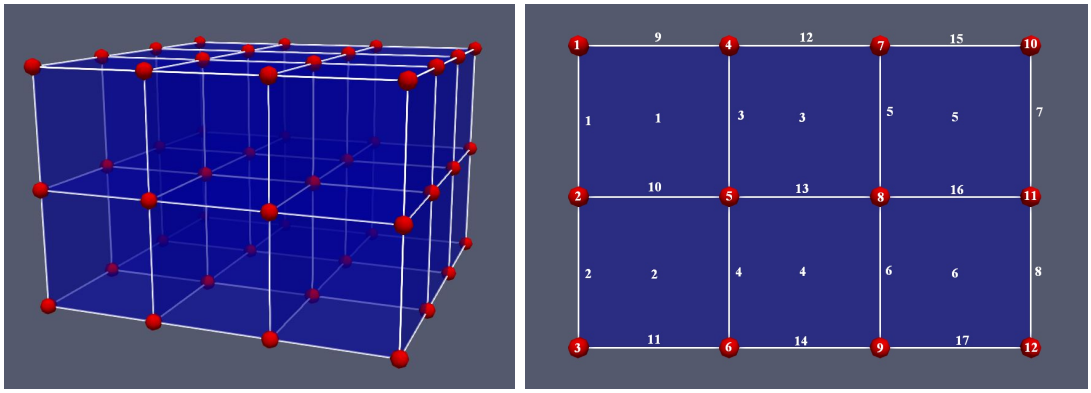

Figure 4.3: [Left]: Cubical complex lattice in 3D. [Right]: Example showing indexing of nodes, edges and faces in 2D. A unique number is specified for each node, edge and face.

from true to false.

- `F` (size: `3 * active nodes`, type: 1D boolean array): This data structure is the faces vector. It stores the primary faces information. This vector indicates whether the faces are active or not, during the collapse operation.

- `C` (size: `active nodes`, type: 1D boolean array): This is the cubes vector. It stores the primary cubes information. This vector is used to indicate whether the cube is active or not. This is the maximum face dimension needed in my application (we have 0-face as the node, 1-face as the edge, 2-face as the face, 3-face as the cube). The length of this vector is the number of the active voxels in the image since only one cube is generated for every voxel.

- `ascending_manifold_label` (size: `3 * active nodes`, type: 1D integer array): The purpose of this vector is to label faces when they are collapsed so that every Morse cell of faces will have the label of the corresponding minima that one will reach if we go in a gradient decent direction from any point on this Morse cell. Mainly, this is used to enable hierarchical simplification to iteratively apply the collapse operation. Hence, I will describe this in more detail in the iterative collapse operation section later. From a data structure point of view, this is a

vector of integers, indicated a label for each minima.

- `edge_faces_link` (size: `2 * 3 * active nodes`, type: 2 * 3 2D integer array): This is used as accelerator accessing faces that are linked to specific edge. During collapse operation, I make a little compromise for a memory usage in favor of speed. Thus, for every edge, I store index information about the neighboring faces. This makes the collapse operation extremely simple and fast. In particular, this allows me to have all edge-face link information ready when visiting a specific edge, which allows me to decide whether it is a free edge or not instantaneously.

- `face_cubes_link` (size: `2 * 3 * active nodes`, type: 2 * 3 2D integer array): Same as the `edge_face_link`, except instead of storing the information about the faces that share a specific edge, I store the cubes sharing a specific face. Thus a face is shared between two cubes. Again, this makes it easy to decide if a face is free or not immediately.

- `edge_weight` (size: `3 * active nodes`, type: 1D float array): This vector is used to store the weight of the edges. The weight is calculated as the average of the weight of the end nodes. A node weight is just the Fast Marching distance in this case. As we will see in the next sections, this vector is sorted in specific order to extract critical structure.

- `face_weight` (size: `3 * active nodes`, type: 1D float array): Same as edge-weight except that the calculated weight is the average of the four nodes since every face has four nodes instead of two in the edge case. This vector is used in the surface extraction algorithm.

- `sorted_indices` (size: `3 * active nodes`, type: 1D integer array): This vector is the output result from sorting edges and faces and contains their indices.

Table 4.1: Summary of data structures for storing and processing the cubical complex.

Name	Description	Size	Type
N0	nodes in image	XSize * YSize * ZSize	1D boolean array
N1	active nodes	active nodes	1D integer array
Type3	face and edges type	3 * active nodes	1D integer array
E	edges	3 * active nodes	1D boolean array
F	faces	3 * active nodes	1D boolean array
C	cubes	active nodes	1D boolean array
faces_label	faces label	3 * active nodes	1D integer array
edge_faces_link	faces sharing edge	2 * 3 * active nodes	2D integer array
face_cubes_link	cubes shring face	2 * 3 * active nodes	2D integer array
edge_weight	edges weight	3 * active nodes	1D float array
face_weight	faces weight	3 * active nodes	1D float array
sorted_indices	sorted indices	3 * active nodes	1D integer array
preserved_list	preserved list	3 * active nodes	1D integer array

As we will see later, `edge_weight` and `face_weight` are sorted as part of the critical structure extraction process. This vector will be used to iterate through edges or faces to apply collapse operations on. The sorting order (ascending or descending) is used to decide which critical structure to extract (generalized maxima or minima).

- `preseved_list` (size: `3 * active nodes`, type: 1D integer array): This vector is filled, while the collapse operation is iterated, to store any preserved (isthmus) edge or face for the next iteration. This is used for both maximal curve and generalized minimal surface extraction since it contains the indices information of edges or faces.

See Table 5.1 for a summary of all these data structures for storing and processing the cubical complex.

4.1.2 Constructing the Cubical Complexes

To construct cubical complex, one has to have two input images. First, a binary image that indicates active and inactive cells (0-face, 1-face,2-face and 3-face). Second, a

cost image to be used for sorting the cubical complex cells. In this thesis, I construct two cubical complexes for maximal curve extraction and one for generalized minimal surface extraction. These are described in the next two-subsections.

Cubical Complex for Maximal Curves of Front: To extract maximal curves, we first need to construct a cubical complex on the Fast Marching front. As we have seen in Chapter 3, a threshold of Fast Marching is extracted in multiple time-step intervals. Every time step is processed to extract the maximal curve on the front. Thus, given a Fast Marching front at a particular time, we want to construct a closed 2D surface for the front, in which we want to locate the highest maximal curve. The steps to build the cubical complex from the Fast Marching front are as follows:

1. Construct a binary image that assigns ones where the geodesic distance is less than or equal to the threshold value and zeros everywhere else.

2. Construct a 3D cubical complex of the binary image by building the data structure and filling them appropriately; this will be a cubical complex of the volume.

3. Now, to construct the closed surface, I keep only faces with its edges that are a subset of two cubes in which one of the cubes has a value of 1 for one of its nodes from the binary image and the other cube has all zero node values from the binary image.

Cubical Complex for Generalized Minima of Fast Marching Distance: To build a cubical complex for generalized minima to extract the surface, we can initially start with the whole image as the active voxels. One could also consider a bounding box that encloses the surface to reduce computational cost. An example of a bounding box would be to use the limits of the boundary curve. Considering a bounding box is useful in cases where the image is very big with respect to a surface. Below, I list the steps and order to fill the data structures for surface extraction:

1. Set active cells in `N0` to true and false otherwise. While setting `N0`, add a reference in `N1` with the index of the cell (in true case only).

2. For every edge (`E`), faces (`F`) and cubes (`C`), check if all nodes are active, then set the cell to true.

3. For every edge, check its active face and reference that in `edge_faces_link`. The same goes for `face_cubes_link`.

4. For `edge_weight` and `face_weight`, calculate the weight for edges and faces as an average of the nodes linked to it.

4.2 Collapse Operation Implementation

I now show how to manipulate the data structures in order to implement the collapse operation. The collapse operation provides an efficient way to simplify a shape, by removing faces while guaranteeing that the operation preserves homotopy equivalence. This is naturally done by checking whether holes are created or destroyed in the cubical complex with a few simple local checks. Note that collapse operation preserves topology and thus guarantees the algorithm correctness in topology. The operation is very simple to implement and requires just local information of neighbors. For example, in the case of maximal curve detection, the only thing to check while iterating through edges is the faces connected to this edge, i.e., faces that share the edge. Algorithm 7 shows how this local check is performed. This requires only checking whether the faces that contain the edge are in the cubical complex. Next, this check is used to perform the collapse operation in Algorithm 8. The algorithm simply checks if the edge is free, and if so, and removes the edge-face pair by setting values in the corresponding indicator arrays to false.

Algorithm 7 IsFree(Edge E) Function Implementation.

```
1: E = edges vector
2: F = faces vector
3: edge_faces_link = accessor for edge neigbours

4: [face1, face2] = edge_faces_link(e)
5: if face1 == true && face2 == true then
6:     return false
7: else if face1 == true then
8:     return face1
9: else if face2 == true then
10:    return face2
11: else
12:    return false
13: end if
```

Algorithm 8 Collapse(Edge e) Operation Implementation

```
1: E = edges vector
2: F = faces vector

3: face=IsFree(edge e)
4: if face ≠ false then
5:     E[e] = false
6:     F[face1] = false
7: end if
```

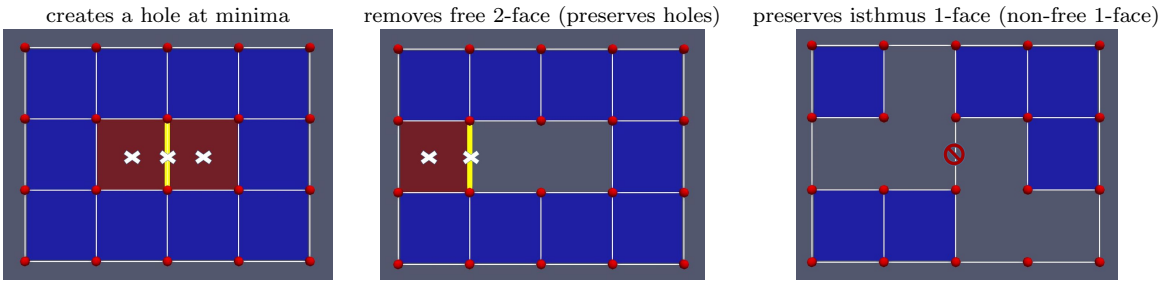

Figure 4.4: Cases encountered when performing the collapse operation for Morse complex extraction. [Left]: The edge is a minima. [Middle]: A free edge. [Right]: An isthmus (non-free) edge.

4.3 Topological Structure Extraction Implementation

In this section, I describe how to extract the Morse complex, simplify it to extract a single maximal curve or a single surface as a generalized minima. I will pay particular attention to how one manipulates the data structures storing the cubical complex to extract the Morse complex and simplify it. Extracting these structures is done by applying the collapse operation on the sorted list of the weighted edges based on Fast Marching as we saw in Chapter 3. I will describe in detail how these structures are obtained from this simple collapse operation in the next sub-sections.

4.3.1 Morse Complex Extraction Implementation

I first describe the implementation of the extraction of the Morse complex. Let us assume we have generated the cubical complex in Figure 4.4. Also, let us assume that we have a cost function on the edges and that edges are sorted from minimum to maximum for the case of maximal curve extraction (generalized minima for surface extraction will be the opposite order). In the case of maximal curve extraction, the goal is to keep any generalized maxima and get rid of minima and regular points. In the midst of performing the collapse operation for Morse complex extraction for extraction of maximal curves, one can observe that one case of the following three cases will occur (see Figure 4.4):

1. **Edge minimum**: In this case, an edge is shared by two faces. This means that the face is at a minimum since the faces are not removed in any earlier collapse operation. This is illustrated in Figure 4.4 (left image). As part of the algorithm, the edge (1-face) and the faces (2-face) that share this edge are removed. As a result, a hole in the cubical complex is created. Note there is no explicit check in the algorithm to detect these minima, but such a case will be encountered as edges are removed from the heap, which is sorted. One can notice that at every local minimum a topological hole will be generated.

2. **Free edge**: In this case, an edge has only one active face connected to it. This will correspond to a regular point (not minima or maxima). Thus the edge and the face are removed, i.e., free pair collapse. This will also not change the topology as it is guaranteed by the collapse operation. This is illustrated in Figure 4.4 (middle image).

3. **Isthmus edge**: In this case, the edge has no face connected to it. The reason that this could occur is that both faces connected to this edge were removed by earlier collapse operations. In this case, the edge will be in the maximal curve structure because all neighboring edges are removed before visiting this edge. Thus, the set of isthmus edges are part of the maximal curves. This is illustrated in Figure 4.4 right image. This edges should be collected in a list, which I call the preserved edge list (`preserved_list` array). Note that the edge is preserved and not removed in the cubical complex.

These three conditions encountered for the collapse operation is part of both maximal curve and generalized minima surface extraction. The pseudo code for maximal curve extraction is presented in Algorithm 9. Note that the algorithm simply checks for each of the three cases, and removes faces and edges in the first two cases, and does nothing but add to the preserved list in the last case.

Algorithm 9 Morse Complex Extraction (Implementation)

```
1: E = edges vector
2: F = faces vector
3: sorted_indices = result of sorting edges
4: edge_faces_link = accessor for edge neigbour
5: preserved_list = list of preserved edges

6: sorted_indices = GenerateEdgeWeightsAndSort()
7: for each e(edge) ∈ sorted_indices do
8:     [face1, face2] = edge_faces_link(e)
9:     if face1 == true && face2 == true then
10:        E[e] = false
11:        F[face1] = false
12:        F[face2] = false
13:    else if face1 == false && face2 == false then
14:        add(e, preserved_list)
15:    else if face1 == false OR face2 == false then
16:        Collapse(e)
17:    end if
18: end for
```

4.3.2 Morse Complex Simplification Implementation

As I have indicated earlier, even though one can extract maximal curves and generalized minima forming the surface by performing the collapse operation, this does not guarantee that this will result in a single curve in maximal curve extraction nor a single closed surface in the case of extracting a closed surface. Thus, one needs to apply the collapse operation iteratively until one ends up with a single curve. Every iteration will produce a simplified Morse complex to be processed again in the next iteration. After each iteration, each Morse cell will be grouped to one face (1-face for edges and 2-face for a face). There is some additional programming effort that is required to group the edges and faces for processing in the next iteration so that the collapse operation can again be applied in the subsequent iterations. In the next sub-sections, I show how to label the ascending or descending manifolds, and then form the simplified complex.

4.3.2.1 Labeling Ascending Manifolds

The key data structure for performing the Morse complex simplification is the `ascending_manifold_l` vector, which is used to label faces. This was described briefly in Section 4.1.1. As described in Section 4.2, there are three cases to consider when performing the collapse operation. I will now describe the labeling of ascending manifolds in each of these three cases.

1. If the **edge is at a minima**, as illustrated in the first case in Figure 4.4 (left image), a new label is generated and both faces sharing this edge are labeled with this newly generated label.

2. In the **free edge** case, as illustrated in Figure 4.4 (middle image), a specific edge is visited and one face containing the edge is not in the complex, but the second face is still in the complex and hence un-labeled. In this case, the label assigned to the face already removed is copied to the unlabeled face.

3. In the **isthmus edge** case, as illustrated in the first case in Figure 4.4 (right image), the edge will be added to the preserved edge list for next iteration, and no action in terms of ascending manifold labeling is performed.

Now, I add this labeling process to Morse complex extraction algorithm to form the updated Algorithm 10.

4.3.2.2 Creating the Simplified Complex

At this stage, I assume that one has computed the first iteration of Morse complex extraction. This includes the labeling described in the previous section. Refer to Figure 3.8 and Figure 3.8 in Chapter 3 for illustrations. At the end of each iteration, a new "super-complex" is constructed to re-run the extraction and further simplify. The operation is referred to as Morse complex *simplification*.

Algorithm 10 Morse Complex Extraction and Labeling (Implementation)

```
1: E = edges vector
2: F = faces vector
3: sorted_indices = result of sorting edges
4: ascending_manifold_label = face label
5: edge_faces_link = accessor for edge neigbour
6: preserved_list = list of preserved edges

7: label_n = 0
8: sorted_indices = GenerateEdgeWeightsAndSort()
9: for each e(edge) ∈ sorted_indices do
10:     [face1, face2] = edge_faces_link(e)
11:     if face1 == true && face2 == true then
12:         E[e] = false
13:         F[face1] = false
14:         F[face2] = false
15:         ascending_manifold_label[face1] = label_n
16:         ascending_manifold_label[face2] = label_n
17:         label_n++
18:     else if face1 == false && face2 == false then
19:         add(e, preserved_list)
20:     else if face1 == false then
21:         Collapse(e)
22:         ascending_manifold_label[face1] = ascending_manifold_label[face2]
23:     else if face2 == false then
24:         Collapse(e)
25:         ascending_manifold_label[face1] = ascending_manifold_label[face2]
26:     end if
27: end for
```

To simplify the Morse complex computed in the previous sub-section, I will create a new complex in which all faces with the same label (belonging to the same ascending manifold) will be considered as one "super-face". Further the edges in the simplified complex will merge all edges of the Morse complex that have the same neighboring faces to a single "super-edge". The previous algorithm for Morse complex extraction is then re-run on the new complex consisting of super-faces and super-edges.

Before one can re-run Morse complex extraction on the simplified "super-complex", one needs to first re-assign weights to the super-edges. The algorithm to do this is shown in Algorithm 11. Algorithm 11 is used at the end of every Morse complex extraction to generate the complex for next iteration.

Algorithm 11 Weighting of Superedges

1: $faces\_group$ = Morse cell of faces
2: $edges\_group$ = Morse cell of edges
3: $edges\_group\_weight$ = average weight of edges in every group
4: $edges\_group\_count$ = total number of edges in every group
5: $preserved\_list$ = list of preserved edges in previous iteration

6: **for each** $e \in preserved\_list$ **do**
7: $id = EdgesGrouping(e)$
8: $edges\_group\_weight[id] += edge\_weight[e]$
9: $edges\_group\_count += 1$
10: **end for**
11: **for** $i = 0$ to size(edges_group_weight) **do**
12: $edges\_group\_weight[i] = \frac{wedges\_group\_weight[i]}{edges\_group\_count[i]}$
13: **end for**

Rather than explicitly constructing a new complex for the "super-complex", I now describe how with the use of some simple data structures, one can implicitly manipulate and keep track of super-faces and super-edges in super-complex when the Morse complex extraction is run on the super-complex. The data structure members will just indicate whether a face group (representing an ascending manifold, i.e., super-face) has been removed in the re-run of Morse complex extraction on the simplified complex. The data structures are:

- `faces_group` (size: `number of labels` , type: 1D boolean array): This vector is used to indicate whether each super-face (a group of faces) has been removed in the Morse complex extraction of the simplified complex. The length of this vector is the number of face labels in the previous iteration of Morse complex extraction, i.e., the number of ascending manifolds. The vector value indicates whether the `faces_group` is active or removed during Morse complex extraction. Initially, all faces will be set to true. The vector index is the ID number of face labels (ascending manifolds) coming from the previous iteration.

- `edges_group` (size: computed as below, type: 1D boolean array): This vector is used to indicate whether each super-edge (a group edges) with same adjacent face labels has been removed or not during Morse complex extraction. The vector value indicates whether an `edges_group` is removed or not. Initially, all `edges_group` will be set to true.

- `edges_group_weight` (size: same as the size of `edges_group`, type: 1D float array): This vector is used to store the weight of the `edges_group`. The weight is calculated as the average of the weight of the edges in `edges_group`.

Now, I introduce the implementation of the complete algorithm for simplification (see Figure 4.5 for an example of the simplification process). The algorithm consists of an iterative loop with two major parts. The first part calls Morse complex extraction, and the second part performs the simplified complex. Note that the Morse complex extraction now also inputs the data structures discussed above (I do not explicitly change the Morse complex routine - Algorithm 10 to deal with these groups, since this is quite obvious). This loop is iterated until only a single boundary curve remains, which means there are only two faces. Since I remove edges one at a time, one will always obtain the case when there are just two faces and one edge, provided that at least two minima exist in the Euclidean minimal path length. This is true assuming

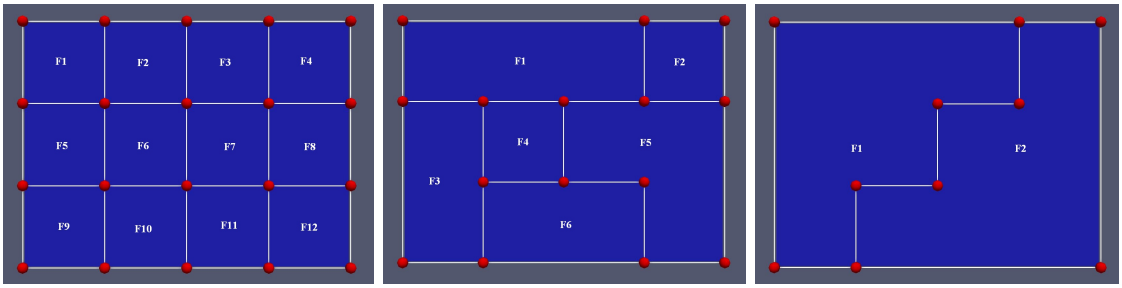

Figure 4.5: Example highest maximal curves extraction of cubical complex (iterative Morse complex simplification). [Left]: Cubical complex in 2D. [Middle]: Maximal curves after first iteration. [Right]: Highest maximal curves in final iteration, and final simplfied complex consisting of two "super-faces" and one "super-edge".

a volcano-like structure that is produced by the Euclidean minimal path distance as discussed in Chapter 3. This is introduced in Algorithm 12.

Algorithm 12 Implementation of Simplification

1: E = edges vector
2: F = faces vector
3: $faces\_group$ = Morse cell of faces
4: $edges\_group$ = Morse cell of edges
5: $preserved\_list$ = list of preserved edges in previous iteration

6: $[preserved\_list, ascending\_manifold\_label]$ =MorseComplexExtraction(E,F)
7: **while** preserved_list contain 3-degree manifold **do**
8: $[preserved\_list, ascending\_manifold\_label]$ =MorseComplexExtraction(edges_group,faces_group)
9: $[edges\_group, faces\_group]$ =SimplifyComplex(preserved_list,ascending_manifold_label)
10: **end while**

4.4 Generalized Minima Surface Extraction Implementation

Based on the fact that a generalized minima is just a generalized maxima of the negative of the function (see Figure 4.6), the algorithm for extracting the generalized minima surface will essentially be identical to that of extracting maximal curves, but in one higher dimension. Thus, instead of processing edges on the heap for maximal curve extraction, one instead places 2-faces on the heap, and instead of removing edge-face pairs, one removes face-cube pairs. Also, the sort of faces will be

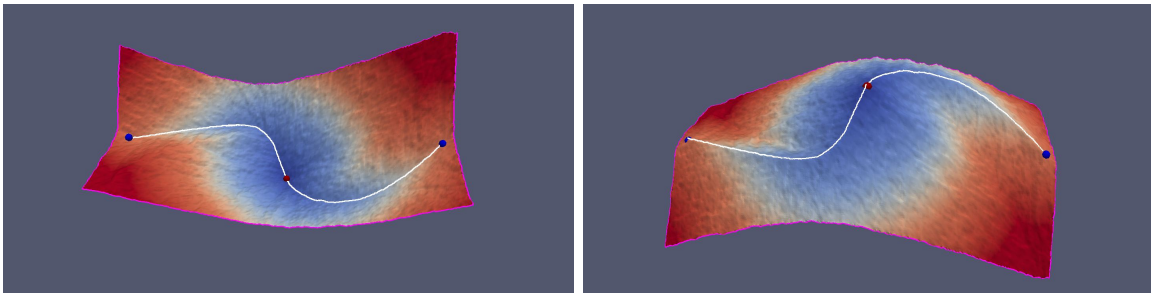

Figure 4.6: Generalized minima of a function are generalized maxima of the negative function. [Left]: Generalized 1-dim minima. [Right]: Generalized 1-dim maxima.

from maximum to minimum rather than sorting edges from minimum to maximum in the case of maximal curve extraction. Moreover, as seen in Chapter 3, for the case of open surface extraction, one does not require any simplification after the first pass of collapse operations. For the case of closed surfaces and cylindrical surfaces, additional passes are required. The last change is that the sorting of faces based on Fast Marching distance, rather than the minimal path Euclidean length. The steps of the algorithm for generalized minima surface extraction are presented in Algorithm 13.

Algorithm 13 Generalized Minima Surface Extraction (Implementation)

F = faces vector
C = cubes vector
$sorted\_indices$ = result of sorting faces

$sorted\_indices = GenerateFacesWeightsAndSort()$
for each $f(face) \in sorted\_indices$ **do**
    face=IsFree(face)
    if $face \neq false$ **then**
        Collapse(face)
    end if
end for

Chapter 5

Experiments and Evaluation

In this chapter, I present the evaluation of my methods both quantitatively and qualitatively and compare with competing methods. I start by introducing the quantitative evaluation methodology. Then, three datasets are presented for the evaluation. The first dataset is a synthetic dataset. The primary goal of the synthetic dataset is to show that my method works for very convoluted surfaces without any compromise to the accuracy of the method. The second dataset was obtained from geophysics domain for extracting fault surfaces from seismic data. The primary goal is to compare my method to the state-of-the-art method in this domain (Crease Surfaces [1]). Finally, the third dataset is from medical image domain. This is to extract fissure surfaces of the lung from CT dataset. Each of these structures will be described briefly in subsequent sub-sections.

5.1 Datasets and Parameters

I evaluate the method on three datasets of 3D images, described below.

Synthetic Dataset: I construct a synthetic dataset consisting of 20 different surfaces with boundary at three different image resolutions, $100 \times 100 \times 100$, $500 \times 500 \times 500$ and $800 \times 800 \times 800$. Each of the surfaces have different 3D boundary curves of different shape, and surfaces that have various degrees of coarse and fine features. Example surfaces are shown in Figure 5.1. The images are formed by setting pixels not within distance 1 to the surface to 1 and all other pixels to 0. The surfaces meshes

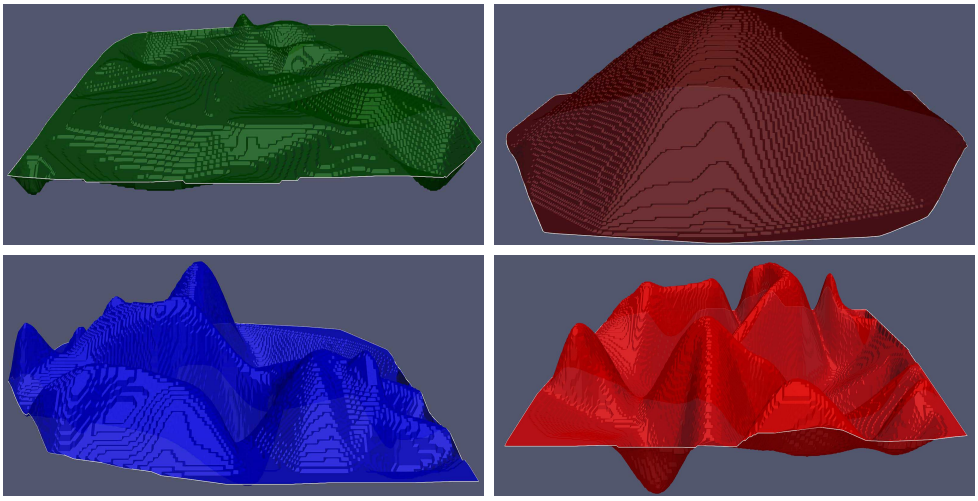

Figure 5.1: Example surfaces in the synthetic dataset. Each surface has a different boundary curve, and the surfaces are of different shape, exhibiting various degrees of randomness.

are downsampled for the lower resolution images. Noise with level $\sigma = 0.1$ is then added to the images.

Seismic Dataset: Seismic images are formed from measurements of seismic pulses reflected back from the earth's sub-surface. They are 3D images, and are used to measure geological structures. I have a dataset of three volumes with dimensions $463 \times 951 \times 651$. The goal is to extract fault surfaces, which form free-boundary surfaces within the volume. Faults may have significant curvature, and the boundaries are non-planar. The images are cluttered and noisy, and faults can be found by locating discontinuities, which is difficult due to subtle edges. Each image consists of multiple faults. I have obtained ground truth segmentations (human annotated) of two faults within each image for each slice.

Lung CT Dataset: I use a dataset of 10 3D computed tomography (CT) of the lung of cancer patients from the Cancer Imaging Archive (TCIA) [97]. Each image has size $512 \times 512 \times Z$, where Z varies between 300 and 700, depending on the patient. The goal is to segment lung fissures (e.g., [98, 99]), which are the boundaries between sections of the lung. They are very thin, subtle structures, and form free-boundary

surfaces. Each of the lung fissures in each image is human annotated, for every slice.

Parameters: The algorithm, given the local surface likelihood ϕ, requires only one parameter, the threshold on the cut cost. In all experiments, I choose this to be $T = 5$. This is not sensitive to the data, and I will show an experiment later for lack of sensitivity to this parameter.

5.2 Evaluation Methodology

I validate the results with quantification measures for both the accuracy of the surface boundary and the surface using quantities analogous to the precision, recall, and F-measure. I represent the surface and its boundary as voxels. Let S_r denote the surface returned by an algorithm and let S_{gt} be the ground truth surface. Denote by ∂S_r and ∂S_{gt} the respective boundaries. I define

$$
\begin{aligned}
N(r \to gt) &= |\{v \in S_r \,:\, d_{S_{gt}}(v) < \varepsilon\}| \\
N(gt \to r) &= |\{v \in S_{gt} \,:\, d_{S_r}(v) < \varepsilon\}| \\
P_S &= N(r \to gt)/|S_r|, \\
R_S &= N(gt \to r)/|S_{gt}|, \\
F_S &= 2P_S R_S/(P_S + R_S) \\
\text{GT Cov.} &= (N(r \to gt) + N(gt \to r))/(|S_r| + |S_{gt}|)
\end{aligned}
$$

where $d_S(v)$ denotes the distance between v and the closet point to S using Euclidean distance, $|\cdot|$ denotes the number of elements of the set, and $\varepsilon > 0$. The precision measures how close the returned surface matches to the ground truth surface. The recall defined above measures how close the ground truth matches to the surface. The F-measure provides a single quantity summarizing both precision and recall. GT-Cov. is another metric summarizing both the precision and recall. All quantities are between 0 and 1 (higher is more accurate). The precision and recall are similar

to accuracy and completness for closed surfaces in evaluating stereo reconstruction algorithms [100]. I similarly define precision $P_{\partial S}$, recall $R_{\partial S}$ and F-measure for ∂S_r and ∂S_{gt} using the same formulas but with the surfaces replaced with their boundaries. I set $\varepsilon = 3$ to account for inaccuracies in the human annotation.

5.3 Evaluation

5.3.1 Synthetic Data: Surface Extraction Given Boundary

I first evaluate three methods for surface extraction given a 3D-boundary curve of the surface, discrete-minimal surface computed with linear programming (LP) [9], discrete-minimal surface approximated with Minimum-Cost Network Flow (MCNF) [9, 46, 101, 102], and the surface extraction, described in Section 3.3. I use Gurobi's state-of-the-art linear programming implementation, to implement LP. I use the Lemon library [103] to implement MCNF. There are no other methods that solve this problem. I choose ϕ to be the image. All methods are provided the ground truth 3D boundary curves. I evaluate the methods in terms of computational time, and in terms of surface accuracy. A summary of results are provided in Table 5.1. Average of results over all the images are provided. My method is computationally faster than all other methods at all resolutions. LP is unable to perform in a reasonable time frame for images sizes above 100^3, and MCNF is unable to perform for image sizes above 500^3. At all resolutions, my method is faster. Speeds are reported on a single Pentium 2.3 GHz processor. The accuracy of my method is also the highest on all measures, but all have similar accuracies. The advantage of my method is clearly speed, and ability to deal with high-resolution images. Note that the analysis was not extended to the real datasets as they have high resolution, making it too computationally expensive to test, and down-sampling the images destroys the structures to be extracted.

Table 5.1: Comparison of methods for surface extraction given the surface boundary on the synthetic dataset. Speed (in seconds), surface precision (P), recall (R), F-measure (F), and ground truth covering (GT-cov) are reported. Higher P, R, F, GT-Cov. indicate better fidelity to the ground truth.

$100 \times 100 \times 100$ pixel images

Method	Time	F	GT-Cov.	P	R
LP	1167	0.93±0.01	0.94±0.01	0.91±0.02	0.96±0.01
MCNF	12.75	0.92±0.01	0.90±0.01	0.93±0.01	0.92±0.02
Surfcut	**1.87**	**0.95±0.02**	**0.95±0.02**	**0.96±0.02**	**0.94±0.03**

$500 \times 500 \times 500$ pixel images

Method	Time	F	GT-Cov.	P	R
LP	>24hr	NA	NA	NA	NA
MCNF	35614	0.94±0.01	0.92±0.01	0.94±0.01	0.93±0.01
Surfcut	**421**	**0.96±0.01**	**0.96±0.01**	**0.97±0.01**	**0.94±0.01**

$800 \times 800 \times 800$ pixel images

Method	Time	F	GT-Cov.	P	R
LP	>24hr	NA	NA	NA	NA
MCNF	>24hr	NA	NA	NA	NA
Surfcut	**2227**	**0.96±0.01**	**0.97±0.01**	**0.98±0.01**	**0.95±0.02**

5.3.2 Seismic Data: Surface and Boundary Extraction

I now compare against the competing method for free boundary surface extraction. To the best of my knowledge, there is no other general algorithm that extracts both the boundary of the free-surface and the surface given a seed point. Therefore, I compare the method in an interactive setting and automated setting (with seed points automatically initialized) to Crease Surfaces [1]. It computes the smoothed Hessian of ϕ, and computes a modified matrix based on the relative difference in the first and second highest eigenvalues. It then forms the surface by determining locations where the eigenvector aligns with the gradient, and constructs connected surfaces. In an interactive setting, I choose the surface returned by [1] that is near to the user-provided seed point (and best fits ground truth) to provide a comparison to my method. In an automated setting, I use a seed point extraction algorithm (described later) to initialize the surface extraction.

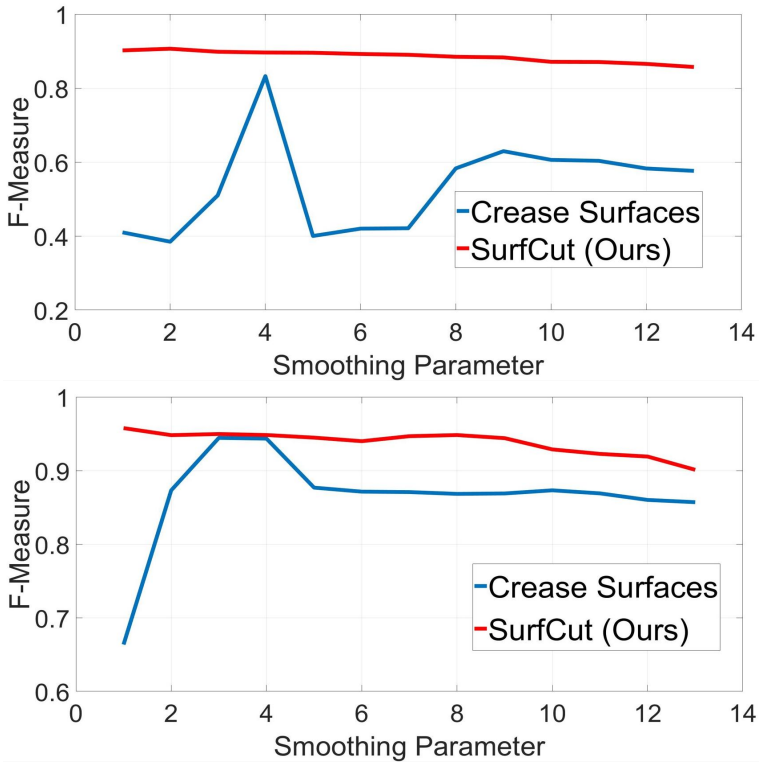

Figure 5.2: **Quantitative Analysis of Smoothing Parameter** Boundary (left) and surface (right) F-measure versus smoothing degradations for my method and [1].

I choose $\phi(x)$ to be the semblance measure in [10]; this along with [1] is state-of-the-art for seismic data.

Robustness to Smoothing Degradations: The semblance ϕ contains a smoothing parameter, which must be tuned to achieve a desirable segmentation. Therefore, it is important that the surface extraction algorithm be robust to changes in the parameter of the likelihood. Thus, I evaluate my algorithm as I vary the smoothing parameter. The smoothing parameter is varied from $\sigma = 0, 2, 3, \ldots, 14$. I initialize my algorithm with a user specified seed point. Quantitative results are shown in Figure 5.2, where I plot the F-measure versus the smoothing amount both in terms of surface and boundary measures. Some visual results of the surfaces are shown in Figure 5.3. Notice my method degrades only gradually and maintains consistently high accuracy in both measures in contrast to [1].

Robustness to Noise: In applications, the image may be distorted by noise

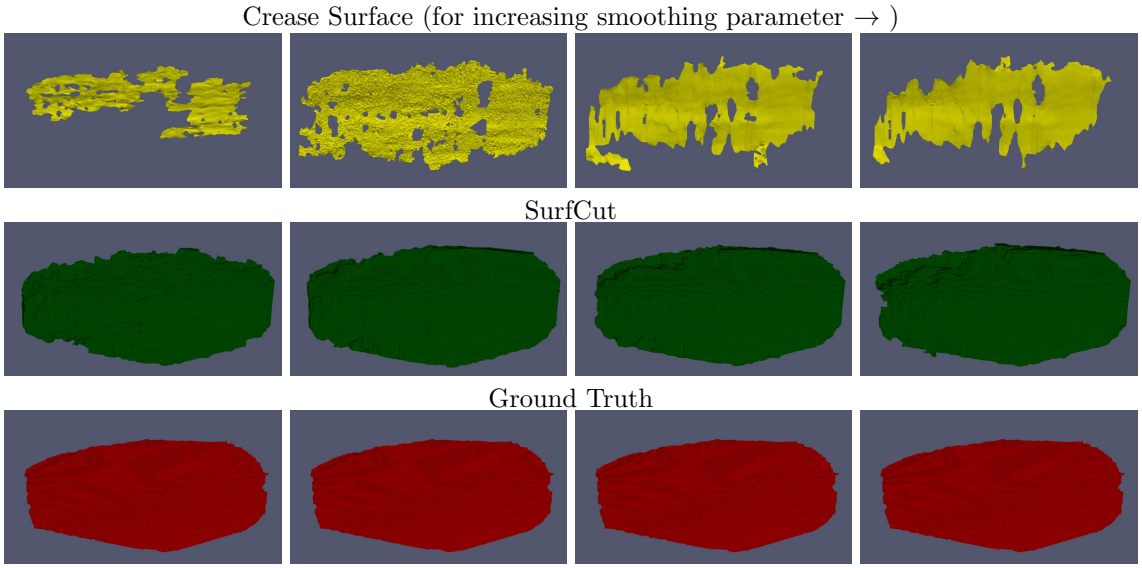

Figure 5.3: **Qualitative Analysis of Smoothing Parameter**. Results displayed by varying the parameter in ϕ (larger towards the right). Surfaces extracted by Crease surfaces and my method are displayed with the ground truth.

(this is the case in seismic images where the SNR may be low), and thus I evaluate my algorithm as I add noise to the image, and I fix the smoothing parameter of the semblance $\phi(x)$ to the one with highest F-measure in the previous experiment. I choose noise levels as follows: $\sigma^2 = 0, 0.05, \ldots, 0.5$. Quantitative results are shown in Figure 5.4, and some visualizations of the surfaces are shown in Figure 5.5. Results show that my method consistently returns an accurate result in both measures, and degrades only slightly.

Slice-wise Validation: I now show some visual validation of my method by showing that the surface intersects with slices of the image in locations where there is a fault, and thus the value of ϕ is low. This is shown in Figure 5.6.

Robustness to Seed-Point Location: I demonstrate that my surface extraction method is robust to the choice of the seed point location. To this end, I randomly sample 30 points (with high local likelihood) from the ground truth surface. I use each of the points as seed points to initialize my algorithm. I measure the boundary and surface accuracy for each of the extracted surfaces. Results are displayed in

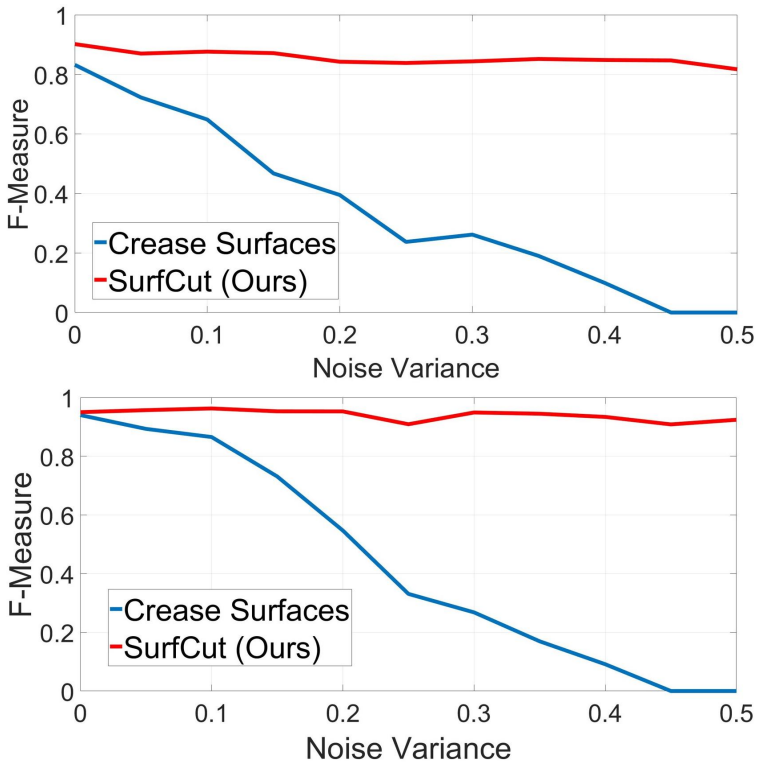

Figure 5.4: **Quantitative Analysis of Noise Degradations** Boundary (left) and surface (right) F-measure versus the noise degradation plots for my method and [1].

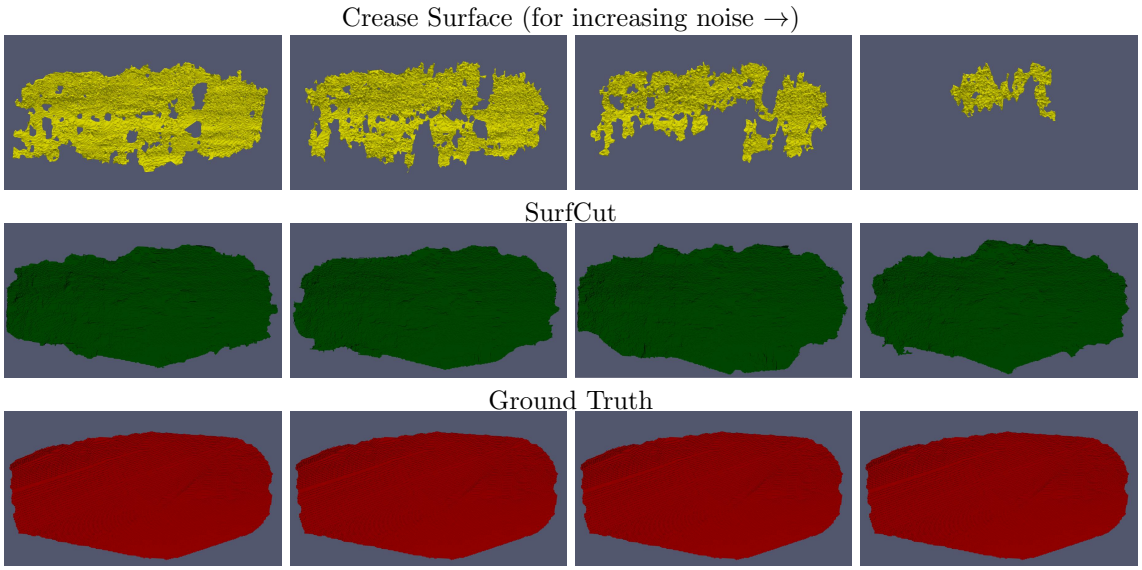

Figure 5.5: **Qualitative Analysis of Noise Degradations**. Results displayed by varying the additive noise to ϕ (larger towards the right). Surfaces extracted by Crease surfaces and my method are displayed with the ground truth.

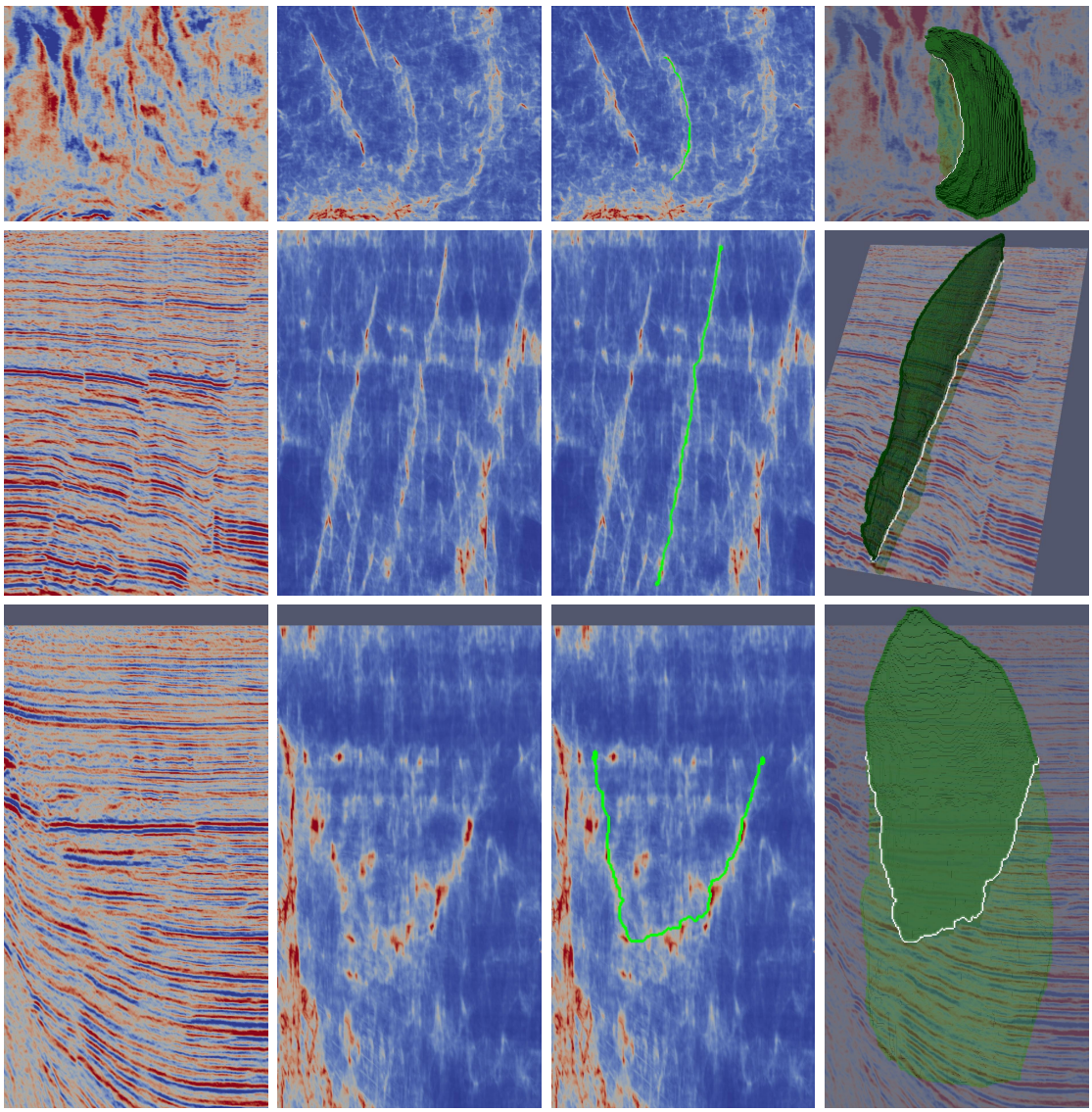

Figure 5.6: **Slice-wise Validation on Seismic Data**: [Left column]: Slices corresponding to x-y, x-z, and y-z planes, [Middle, left]: Local surface likelihood ($1/\phi$), [Middle right]: Intersection of SurfCut result (green) with slice, [Right column]: surface from SurfCut at certain viewpoint.

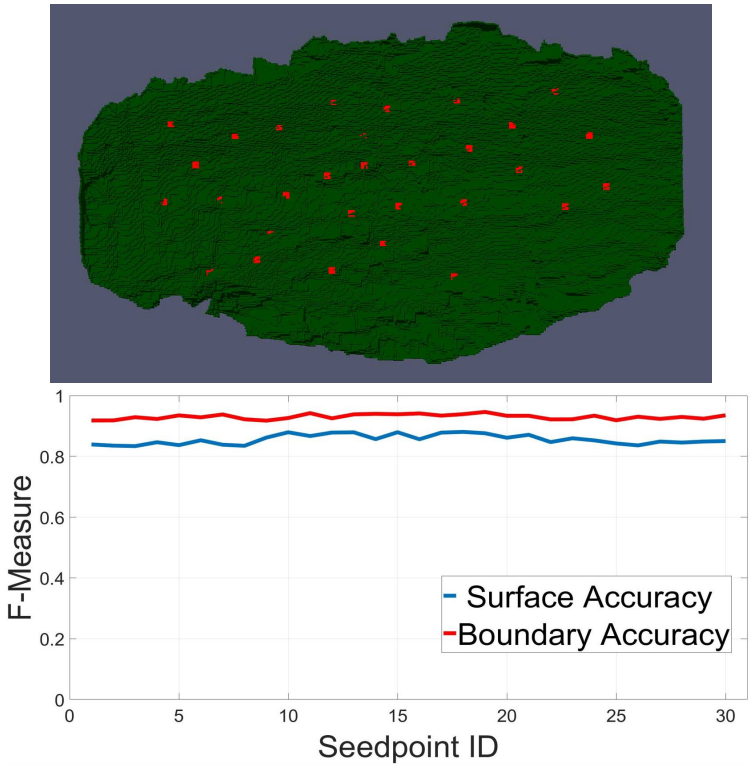

Figure 5.7: **Robustness to Seed Point Choice**: [Left]: A visualization of the seed points chosen. [Right]: Boundary F-measure versus various seed point indices. The same boundary and surface accuracy is maintained no matter the seed point location.

Figure 5.7. They show my algorithms consistently returns a boundary and surface of similar accuracy regardless of the seed point location.

Robustness to Seed-Point Location: I analyze the sensitivity of the parameter of my algorithm - the threshold on the average cut cost in the surface boundary extraction. To this end, I run my algorithm with varying thresholds from $T = 0, 1, \ldots, 20$, and then measure the accuracy of the boundary curve and then the extracted surface. The results are displayed in Figure 5.8. This shows that a wide range of thresholds produces nearly identical results.

Robustness to CUSUM Threshold: In practice, the threshold for the CUSUM algorithm for terminating the front propagation should be relative to the quantity ΔD, the increment of the threshold for the distance function U in Section 3.2. In particular, larger values imply that the distance between adjacent curves would be

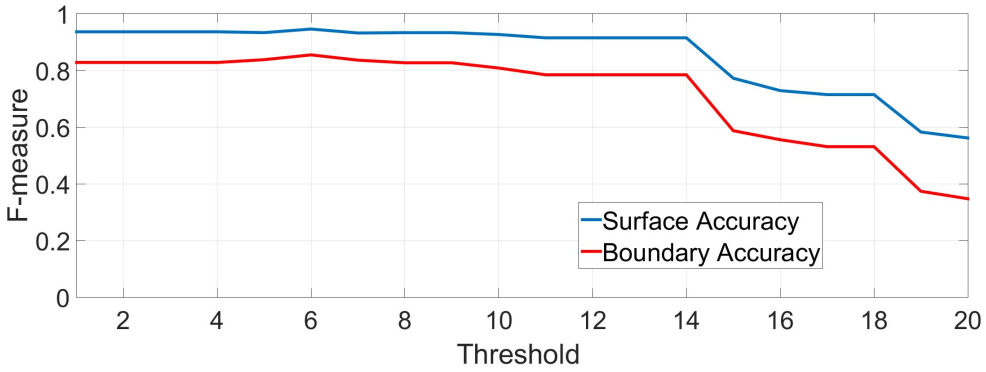

Figure 5.8: **Quantitative Analysis of Sensitivity of Threshold** This analyzes the sensitivity of the boundary curve extraction to the threshold on the average cut cost to stop the algorithm. The accuracy of the boundary and surface measured by F-measure versus various thresholds is plotted.

larger. The exact relation comes from the eikonal equation, which can be written as $\frac{\partial U}{\partial N} = \phi$, where N is the outward normal. This can be used to find $\Delta D_E = \Delta D/\phi$, where ΔD_E is the Euclidean distance between adjacent curves. Therefore, in practice, the threshold should be set $T \in \Delta D \times [1/\phi_{max}, 1/\phi_{min}]$, where ϕ_{min} (ϕ_{max}) is the minimum (maximum) value of ϕ. The exact value would need to be set via a training set and procedure. Figure 5.8 shows the algorithm functions well for a thresholds in a wide range.

Analysis of Automated Algorithm: Even though the contribution is in the surface and boundary extraction from a seed point, I show with a seed point initialization, the method can be automated. I initialize the algorithm with a simple automated detection of seeds points. I extract seed points by finding extrema of the Hessian and then running a piece-wise planar segmentation of these points using RANSAC [104] successively; the point on each of the segments located closest to other points on the segment are seed points. This operates under the assumption that the surfaces are roughly planar. If not, there could possibly be redundant seed points on the same surface, which would result in repetitions in surfaces in my final output. This could easily be filtered out. I run the boundary curve extraction followed by sur-

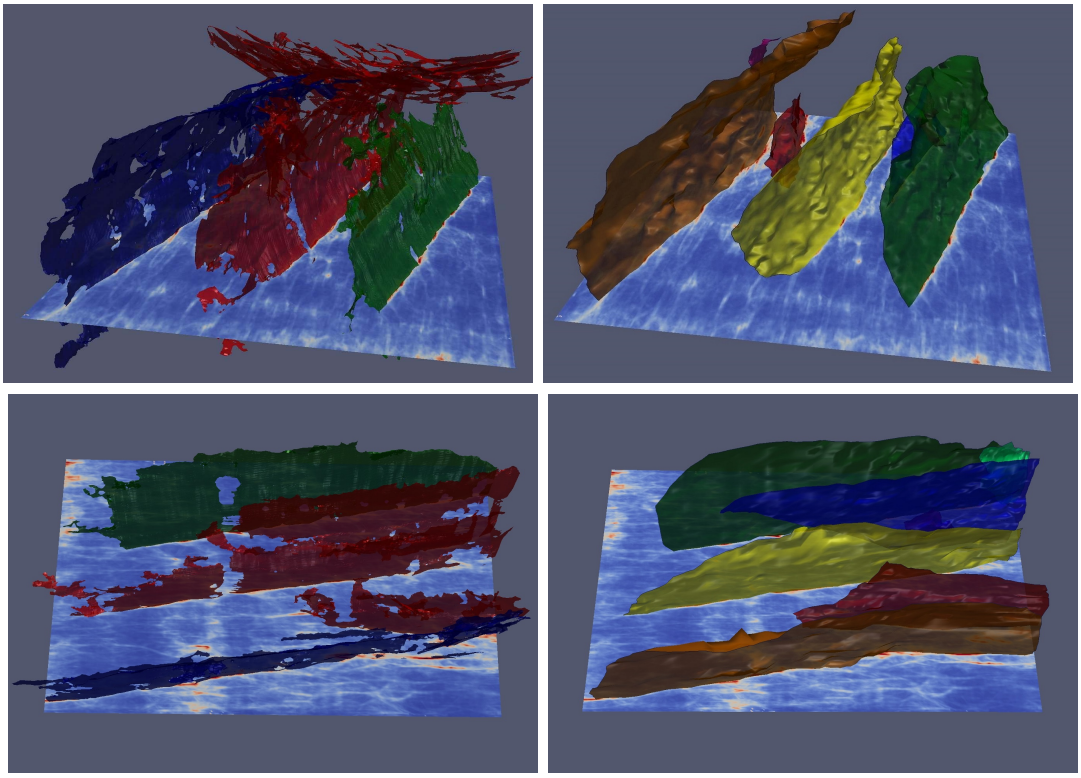

Figure 5.9: Example result in an automated setting. [Left]: Result by Crease Surfaces, which contains holes and incorrectly detects clutter (top, red) due to noise in the data. [Right]: Results of SurfCut, which extracts the correct number of surfaces and produces smooth simple surfaces.

face extraction for each of the seed points on the original datasets. I compare to [1]. There are 6 ground truth surfaces in this dataset. The algorithm correctly extracts 6 surfaces, while [1] extracts 4 surfaces (2 pairs of faults are merged together each as a single connected component). Results on a dataset are visualized in Figure 5.9 (each connected component in different color). Notice that Crease Surfaces has holes, captures clutter, and connects separate faults.

Computational Cost of Automated Algorithm: I analyze run-times on a dataset of size $463 \times 951 \times 651$. The run-time of the algorithm depends on the size of the surface. To extract one surface, the algorithm takes on average 10 minutes (9 minutes for the boundary extraction and 1 minute for the surface extraction). Automated seed point extraction takes about 3 minutes. Therefore, the total cost

of the algorithm for extracting 6 faults is about 1 hour. I note that after seed point extraction, the computation of surfaces can be parallelized. In comparison, [1] takes about 2 hours on the same dataset. Speeds are reported on a single Pentium 2.3 GHz processor. The accuracy of my method is also the highest on all measures, but all have similar accuracies. The advantage of my method is clearly speed, and ability to deal with high-resolution images. Note that the analysis was not extended to the real datasets as they have high resolution, making it too computationally expensive to test, and down-sampling the images destroys the structures to be extracted.

5.3.3 Lung CT Data: Surface and Boundary Extraction

I now compare to Crease Surfaces for the Lung CT dataset. I compare the methods under the settings described in the previous section. For medical data, I modify the matrix based on the Hessian in Crease Surfaces to another matrix based on closeness to a plate-like structure as common in lung fissure detection [105, 106, 99]. State-of-the-art methods in fissure extraction use a method similar to Crease surfaces to extract the surface. I choose ϕ to be the plate-ness measure in the method. Quantitative results on the entire dataset are summarized in Table 5.2. Both in terms of surface and boundary accuracy, the method is more accurate with respect to all measures. Visual validation of the method on slice-wise views of the surface and image is shown in Figure 5.10. Some visualizations of the surface results are shown in Figure 5.11. Various slices are shown to help visualize features of the image. Crease surface generates surfaces with incorrect holes and many times cannot capture the entire fissure, hence low recall and precision on the boundary metrics. SurfCut does not contain any holes and accurately captures very fine and thin structures near the boundaries of the fissures.

Table 5.2: **Quantitative Evaluation on Lung Dataset**. Comparison of methods in terms of surface and boundary accuracy. Precision (P), recall (R), F-measure (F), and ground truth covering (GT-cov) are reported. Higher P, R, F, GT-Cov. indicate better fidelity to the ground truth.

Surface accuracy

Method	F	GT-Cov.	P	R
Crease Surfaces	0.76±0.08	0.70±0.10	0.67±0.11	0.91±0.06
Surfcut	**0.91±0.04**	**0.87±0.06**	**0.86±0.06**	**0.95±0.02**

Boundary accuracy

Method	F	GT-Cov.	P	R
Crease Surfaces	0.70±0.11	0.72±0.08	0.69±0.10	0.71±0.12
Surfcut	**0.86±0.04**	**0.86±0.06**	**0.85±0.06**	**0.87±0.05**

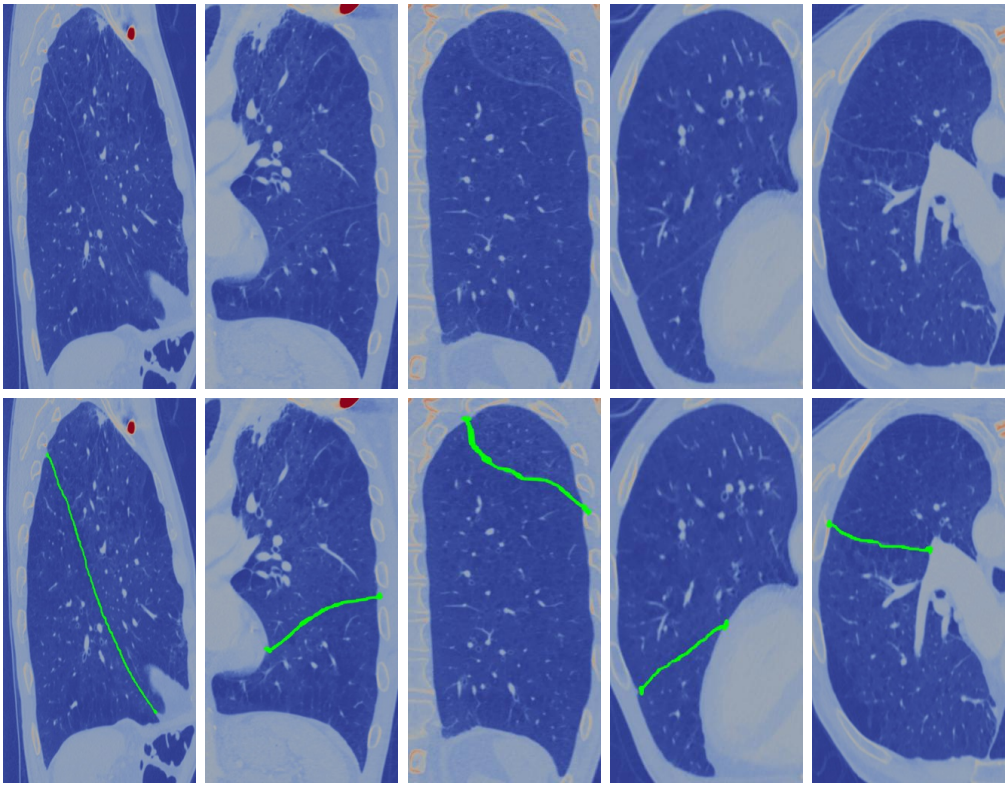

Figure 5.10: **Slice-wise Validation in Lung CT Dataset**. [Top]: Various slices of an image of a patient, [Bottom]: Surface generated with SurfCut intersected with the slice above (green) superimposed on the slice. Notice the structure of interest is a subtle thin lines in the slices (top).

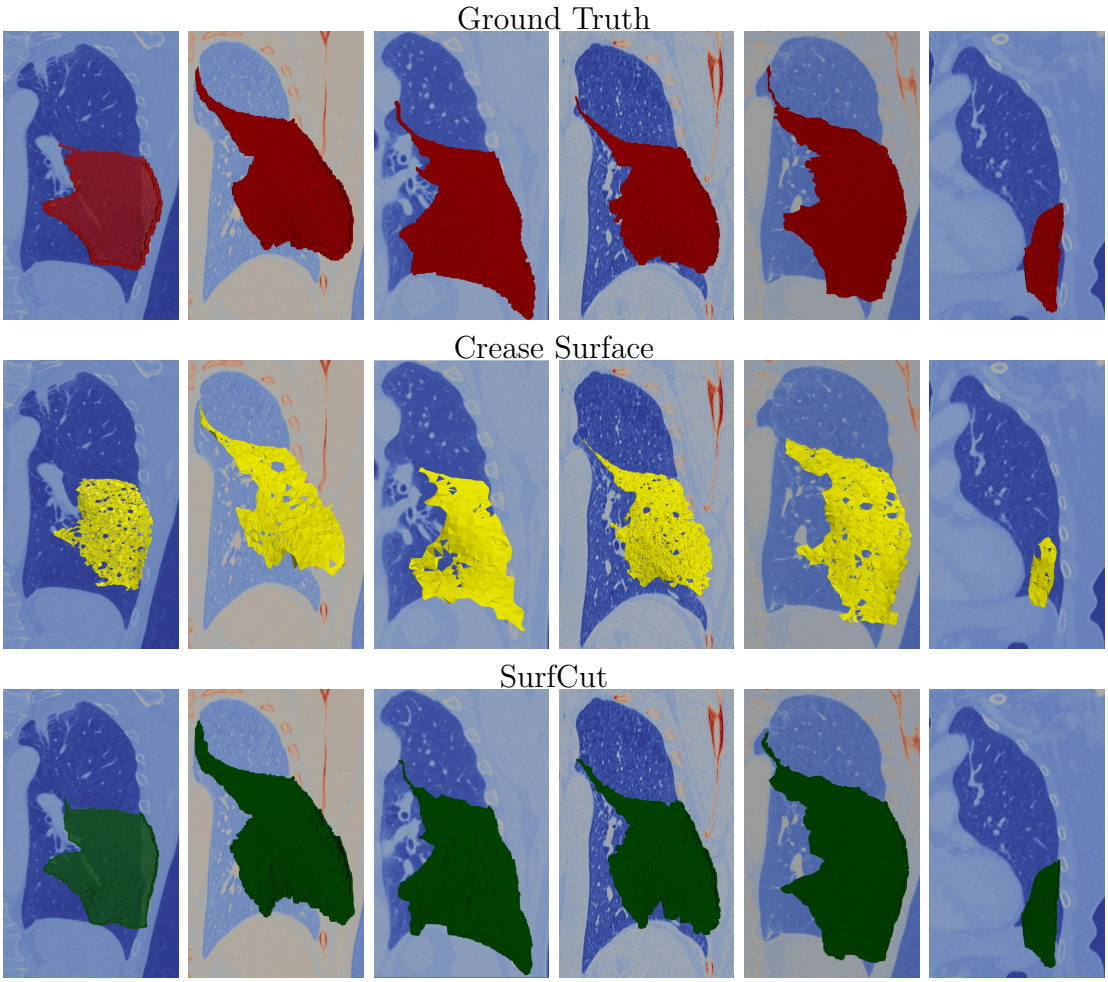

Figure 5.11: **Qualitative Results on Lung CT**. Columns show the surfaces on the same slice on the same patient for various methods. Moving through a row shows the surface for different patients and a slice of the image is shown for various different slices. SurfCut extracts more of the fine structure of the fissures, better estimates the boundary and recovers more of the surfaces than Crease surfaces.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

A general method for extracting a smooth and simple (without holes) surface with unknown boundary in a 3D image with noisy local measurements of the surface, e.g., edges has been introduced. The novel method takes as input a single seed point and extracts the unknown boundary that may lie in 3D. It then uses this boundary curve to determine the entire surface efficiently. I have demonstrated with extensive experiments on noisy and corrupted data with possible interruptions that the method accurately determines both the boundary and the surface, and the method is robust to seed point choice. In comparison to an approach which extracts connected components of edges in 3D images, the method is more accurate in both surface and boundary measures. The computational cost of the algorithm is less than competing approaches. The specific contributions of this thesis are listed in the following paragraphs.

The first part of the algorithm extracts the boundary curve of open surfaces given only a seed point selected by the user anywhere on the surface. This is In contrast to all other globally optimal methods for open surface extraction that require a full boundary curve as input. This is very tedious in practice to input by hand. Input of only one seed point is a great advancement of the literature. Furthermore, the seed point extraction can be automated by algorithms that are in development as future work. The proposed method is the first method that can extract the boundary curve

automatically given a seed point as input.

Given the boundary curve which is automatically extracted, I introduced an efficient algorithm to obtain a free-boundary surface. Unlike all other globally optimal methods, the method has shown an unprecedented result in terms of speed without any sacrifices to accuracy. In fact, other globally optimal methods suffer greatly from slow computation time especially as the data grows. As was shown in experiments, the computation time increases exponentially for those prior methods, which is not the case for the method presented in this thesis.

The developed methods are based on techniques in minimal path theory, the Morse theory, and computational topology. These theories allow for the design of a principled algorithm which is robust to noise and can handle complex topology. Specifically, I have exploited the Morse complex to extract both curves and surfaces from 3D volumetric data. I have also made use of cubical complexes to guarantee topological properties of the desired solution. To the best of my knowledge, this is the first work to develop a mathematical framework for surface extraction using concepts from Morse theory.

Moreover, I have conducted extensive experiments to show the advantages of my methods. I have demonstrated that my method is the most practical choice in comparison to minimal surface approaches. Particularly, I compare to minimal surface formulated as a linear program and also as an MCNF (minimum cost flow networks). I have shown much better results for computation time and accuracy. Also, I have tested my method in two important real applications. First, for fault surface extraction from 3D seismic data. I have compared my method against the state of the art method in this domain, namely, crease surface extraction method. I have shown better results in terms of the accuracy of both boundary and the surface. For the second application from medical images, I have compared my method to Crease Surfaces for lung fissure extraction in CT data. I also have demonstrated a better result in this

domain.

Furthermore, I have shown that the framework developed is general and not limited to specific surface topologies. It applies to multiple topologies such as open surfaces, closed surfaces and cylindrical topology surfaces (unlike other methods, which only deal with a particular topology and the methods each have different a mathematical framework). Moreover, the methods apply to other imaging modalities as well. Throughout this thesis, I have shown several examples from geophysical 3D seismic data and medical images such as CT and MRI.

6.2 Future Work

Although the main focus of the thesis has been free-boundary surface extraction, I did show the promise of the methodologies to other surface topologies. I have shown some preliminary results of applying the current algorithms to other topologies such as closed surface and cylindrical topology on synthetic data. In future, one could apply the algorithms to real data examples in medical applications and other domains and to compare with competing approaches in the respective field. I believe there would be a great advantage over Graph Cuts in terms of speed, and preliminary experiments indicate that my methodology is more robust with respect to the the attribute choice.

In the 3D seismic image interpretation area of applications, there are several geological structures to extract using the current framework such as salt diapers, horizons, channels, etc. In fact, this is a very active research area for detecting and extracting such geological structures due to various challenges such as signal to noise level. An accurate algorithm to do so is a long-standing problem for many of the structurally and stratigraphically complex-to-interpret seismic structures. Since the method is general enough to deal with different topologies and robust enough to handle noise, one could use and customize the current framework to extract these geological structures.

Moreover, an essential ingredient of object extraction system is to provide interactive tools to enable editing and adjusting results to the desired level of accuracy. An advantage of this framework is that it is straightforward to edit the data by adding additional seed points to drive the surface to go through pre-specified locations. Therefore, another area for future work is to develop efficient tools to make a users life easier and surface extraction a rapid process. Currently, users both in the domain of medicine and seismic interpretation would greatly appreciate having surface segmentation tools that with a few mouse clicks could obtain the desired system. It is believed that fully automated tools are still too lofty a goal, and thus a convenient user interactive approach would certainly bring tremendous progress to those two fields.

Currently, the algorithm requires a seed point as an input for each surface. An area of future work is to automate the generation of seed points so that it can be used as a fully automated system to extract open surfaces from images such as seismic images. Current tools do not provide a robust method to find seed points only belonging to a single surface such that each surface will have a single seed point chosen and no surface is missing. Therefore, development of automated seed point extraction is certainly an area for future work. I have shown a possible technique in which one can build from in the previous chapter.

In this thesis, I focused on structured data (3D images). The algorithm might have useful applications in non-structured data such as point clouds for surface extraction. Moreover, the algorithm can be explored for mesh processing and segmentation applications as non-structured data. Many of the algorithms and techniques presented in this thesis generalize to mesh-data, such as surface reconstruction.

The method also applies to the detection of curves in 2D images, as the algorithm applies to critical structures in any dimension. Since the method produces smooth curves, I believe it would improve edge detection, which is a great challenge in com-

puter vision. Existing algorithms produce gaps in edges, which is problematic since it is difficult to form a segmentation with gaps. The algorithm is robust to interruptions produced by fine features in images. Future work could be to see if the method is feasible in this area. Other methods tend to operate in a heuristic manner that leads to missing some structures for example in edge extraction in 2D. If a robust cost function is employed and a reasonable criterion for an iterative algorithm is designed, it might result in a better detection due to computational topology used.

In this thesis, I have used the Fast Marching algorithm to compute geodesic distance. This acts mostly as a cost function which turns out to be robust and fast. There are recent developments for computing Fast Marching with higher speed but many times with approximate results. Also, higher accuracy Fast Marching is developed for anisotropic cases. These methods are worthy of further investigation as a better output of Fast Marching improves both computation and accuracy of the developed method. There are also other distance transforms that may not be geodesic distance that might be worth trying out.

REFERENCES

[1] T. Schultz, H. Theisel, and H.-P. Seidel, "Crease surfaces: From theory to extraction and application to diffusion tensor mri," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 1, pp. 109–119, 2010.

[2] L. D. Cohen and R. Kimmel, "Global minimum for active contour models: A minimal path approach," *International journal of computer vision*, vol. 24, no. 1, pp. 57–78, 1997.

[3] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[4] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1528–1538, 1995.

[5] V. Kaul, A. Yezzi, and Y. Tsai, "Detecting curves with unknown endpoints and arbitrary topology using minimal paths," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 10, pp. 1952–1965, 2012.

[6] J. Mille, S. Bougleux, and L. D. Cohen, "Combination of piecewise-geodesic paths for interactive segmentation," *International Journal of Computer Vision*, vol. 112, no. 1, pp. 1–22, 2015.

[7] F. Benmansour and L. D. Cohen, "From a single point to a surface patch by growing minimal paths," in *Scale Space and Variational Methods in Computer Vision*. Springer, 2009, pp. 648–659.

[8] R. Ardon, L. D. Cohen, and A. Yezzi, "A new implicit method for surface segmentation by minimal paths: Applications in 3d medical images," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, 2005, pp. 520–535.

[9] L. Grady, "Minimal surfaces extend shortest path segmentation methods to 3d," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 2, pp. 321–334, 2010.

[10] D. Hale, "Methods to compute fault images, extract fault surfaces, and estimate fault throws from 3d seismic images," *Geophysics*, vol. 78, no. 2, pp. O33–O43, 2013.

[11] J. M. Sullivan, "A crystalline approximation theorem for hypersurfaces," *Princeton Ph. D*, 1990.

[12] D. Hale *et al.*, "Fault surfaces and fault throws from 3d seismic images," in *2012 SEG Annual Meeting*. Society of Exploration Geophysicists, 2012.

[13] N. Paragios and R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 3, pp. 266–280, 2000.

[14] C. Li, C. Xu, C. Gui, and M. D. Fox, "Level set evolution without re-initialization: a new variational formulation," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 430–436.

[15] M. Y. Wang, X. Wang, and D. Guo, "A level set method for structural topology optimization," *Computer methods in applied mechanics and engineering*, vol. 192, no. 1, pp. 227–246, 2003.

[16] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, "A hybrid particle level set method for improved interface capturing," *Journal of Computational physics*, vol. 183, no. 1, pp. 83–116, 2002.

[17] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[18] N. Paragios and R. Deriche, "Geodesic active regions and level set methods for supervised texture segmentation," *International Journal of Computer Vision*, vol. 46, no. 3, pp. 223–247, 2002.

[19] D. Adalsteinsson and J. A. Sethian, "The fast construction of extension velocities in level set methods," *Journal of Computational Physics*, vol. 148, no. 1, pp. 2–22, 1999.

[20] J. A. Sethian and A. Wiegmann, "Structural boundary design via level set and immersed interface methods," *Journal of computational physics*, vol. 163, no. 2, pp. 489–528, 2000.

[21] M. Rousson and N. Paragios, "Shape priors for level set representations," *Computer VisionECCV 2002*, pp. 416–418, 2002.

[22] O. Faugeras and R. Keriven, *Variational principles, surface evolution, PDE's, level set methods and the stereo problem.* IEEE, 2002.

[23] B. J. Kadlec, G. A. Dorn, H. M. Tufo, and D. A. Yuen, "Interactive 3-d computation of fault surfaces using level sets," *Visual Geosciences*, vol. 13, no. 1, pp. 133–138, 2008.

[24] J. Dong and C. Hao, "3d fast level set image segmentation based on chan-vese model," in *2009 3rd International Conference on Bioinformatics and Biomedical Engineering.* IEEE, 2009, pp. 1–4.

[25] R. Kimmel, D. Shaked, N. Kiryati, and A. M. Bruckstein, "Skeletonization via distance maps and level sets," *Computer vision and image understanding*, vol. 62, no. 3, pp. 382–391, 1995.

[26] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[27] G. Sundaramoorthi, A. Yezzi, and A. C. Mennucci, "Sobolev active contours," *International Journal of Computer Vision*, vol. 73, no. 3, pp. 345–366, 2007.

[28] G. Sundaramoorthi, A. Yezzi, and A. Mennucci, "Coarse-to-fine segmentation and tracking using sobolev active contours," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 5, pp. 851–864, 2008.

[29] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations," *Journal of computational physics*, vol. 79, no. 1, pp. 12–49, 1988.

[30] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[31] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International journal of computer vision*, vol. 22, no. 1, pp. 61–79, 1997.

[32] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[33] J. Ulen, P. Strandmark, and F. Kahl, "Shortest paths with higher-order regularization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 12, pp. 2588–2600, 2015.

[34] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[35] ——, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[36] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.

[37] Y. Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in nd images," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1. IEEE, 2001, pp. 105–112.

[38] P. Kovács, "Minimum-cost flow algorithms: an experimental evaluation," *Optimization Methods and Software*, vol. 30, no. 1, pp. 94–127, 2015.

[39] M. H. Narkhede, "A review on graph based segmentation."

[40] I. Oguz and M. Sonka, "Logismos-b: layered optimal graph image segmentation of multiple objects and surfaces for the brain," *Medical Imaging, IEEE Transactions on*, vol. 33, no. 6, pp. 1220–1235, 2014.

[41] ——, "Logismos-b: layered optimal graph image segmentation of multiple objects and surfaces for the brain," *Medical Imaging, IEEE Transactions on*, vol. 33, no. 6, pp. 1220–1235, 2014.

[42] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," in *ACM transactions on graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 309–314.

[43] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear inequalities and related systems*, vol. 38, pp. 225–231, 2003.

[44] L. R. Ford and D. Fulkerson, *Flow in networks*. Princeton University Press, 1962.

[45] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan, "Finding minimum-cost flows by double scaling," *Mathematical programming*, vol. 53, no. 1, pp. 243–266, 1992.

[46] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *Journal of algorithms*, vol. 22, no. 1, pp. 1–29, 1997.

[47] Z. Király and P. Kovács, "Efficient implementations of minimum-cost flow algorithms," *arXiv preprint arXiv:1207.6381*, 2012.

[48] ——, "Efficient implementations of minimum-cost flow algorithms," *arXiv preprint arXiv:1207.6381*, 2012.

[49] A. Sifaleras, "Minimum cost network flows: Problems, algorithms, and software," *Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334-6043*, vol. 23, no. 1, 2013.

[50] L. Grady, "Computing exact discrete minimal surfaces: Extending and solving the shortest path problem in 3d with application to segmentation," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 69–78.

[51] T. Hollt, J. Beyer, F. Gschwantner, P. Muigg, H. Doleisch, G. Heinemann, and M. Hadwiger, "Interactive seismic interpretation with piecewise global energy minimization," in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*. IEEE, 2011, pp. 59–66.

[52] ——, "Interactive seismic interpretation with piecewise global energy minimization," in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*. IEEE, 2011, pp. 59–66.

[53] D. Patel, S. Bruckner, I. Viola, and E. M. Gröller, "Seismic volume visualization for horizon extraction," in *Visualization Symposium (PacificVis), 2010 IEEE Pacific*. IEEE, 2010, pp. 73–80.

[54] R. Amorim, E. V. Brazil, D. Patel, and M. C. Sousa, "Sketch modeling of seismic horizons from uncertainty," in *Proceedings of the International Symposium on Sketch-based interfaces and modeling*. Eurographics Association, 2012, pp. 1–10.

[55] N. Keskes, P. Zaccagnino, D. Rether, and P. Mermey, "Automatic extraction of 3-d seismic horizons," in *SEG Technical Program Expanded Abstracts 1983*. Society of Exploration Geophysicists, 1983, pp. 557–559.

[56] H. Borgos, T. Skov, and L. Sønneland, "Automated structural interpretation through classification of seismic horizons," *Mathematical Methods and Modelling in Hydrocarbon Exploration and Production*, pp. 89–106, 2005.

[57] H. G. Borgos, O. Gramstad, G. V. Dahl, P. Le Guern, L. Sonneland, and J. F. Rosalba, "Extracting horizon patches and geo-bodies from 3d seismic waveform sequences," in *SEG Technical Program Expanded Abstracts 2006*. Society of Exploration Geophysicists, 2006, pp. 1595–1599.

[58] J. Hoyes and T. Cheret, "A review of global? interpretation methods for automated 3d horizon picking," *The Leading Edge*, vol. 30, no. 1, pp. 38–47, 2011.

[59] P. de Groot, A. Huck, G. de Bruin, N. Hemstra, and J. Bedford, "The horizon cube: A step change in seismic interpretation!" *The Leading Edge*, vol. 29, no. 9, pp. 1048–1055, 2010.

[60] S. I. Pedersen, T. Randen, L. Sonneland, and Ø. Steen, "Automatic fault extraction using artificial ants," in *SEG Technical Program Expanded Abstracts 2002*. Society of Exploration Geophysicists, 2002, pp. 512–515.

[61] W.-K. Jeong, R. Whitaker, and M. Dobin, "Interactive 3d seismic fault detection on the graphics hardware," 2006.

[62] D. B. Neff, J. R. Grismore, and W. A. Lucas, "Automated seismic fault detection and picking," Jan. 25 2000, uS Patent 6,018,498.

[63] S. I. Pedersen, T. Skov, A. Hetlelid, P. Fayemendy, T. Randen, and L. Sønneland, "New paradigm of fault interpretation," in *SEG Technical Program Expanded Abstracts 2003*. Society of Exploration Geophysicists, 2003, pp. 350–353.

[64] P. Jacquemin and J.-L. Mallet, "Automatic faults extraction using double hough transform," in *SEG Technical Program Expanded Abstracts 2005*. Society of Exploration Geophysicists, 2005, pp. 755–758.

[65] X. Zhou, T. Hayashi, T. Hara, H. Fujita, R. Yokoyama, T. Kiryu, and H. Hoshi, "Automatic recognition of lung lobes and fissures from multislice ct images," in *Proceedings of SPIE*, vol. 5370, 2004, pp. 1629–1633.

[66] D. Wu, L. Lu, J. Bi, Y. Shinagawa, K. Boyer, A. Krishnan, and M. Salganicoff, "Stratified learning of local anatomical context for lung nodules in ct images," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2791–2798.

[67] A. Yıldız, F. Gölpınar, M. Çalıkoğlu, M. N. Duce, C. Özer, and F. D. Apaydın, "Hrct evaluation of the accessory fissures of the lung," *European journal of radiology*, vol. 49, no. 3, pp. 245–249, 2004.

[68] L. Zhang, E. A. Hoffman, and J. M. Reinhardt, "Atlas-driven lung lobe segmentation in volumetric x-ray ct images," *IEEE transactions on medical imaging*, vol. 25, no. 1, pp. 1–16, 2006.

[69] S. Ukil and J. M. Reinhardt, "Anatomy-guided lung lobe segmentation in x-ray ct images," *IEEE transactions on medical imaging*, vol. 28, no. 2, pp. 202–214, 2009.

[70] J. G. Tamez-Pena, M. Barbu-McInnis, and S. Totterman, "Knee cartilage extraction and bone-cartilage interface analysis from 3d mri data sets," in *Medical Imaging 2004: Image Processing*, vol. 5370. International Society for Optics and Photonics, 2004, pp. 1774–1785.

[71] K. Zhang, J. Deng, and W. Lu, "Segmenting human knee cartilage automatically from multi-contrast mr images using support vector machines and discriminative random fields," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE, 2011, pp. 721–724.

[72] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, "Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network," in *International conference on medical image computing and computer-assisted intervention*. Springer, 2013, pp. 246–253.

[73] L. Zhu, Y. Gao, V. Appia, A. Yezzi, C. Arepalli, T. Faber, A. Stillman, and A. Tannenbaum, "A complete system for automatic extraction of left ventricular myocardium from ct images using shape segmentation and contour evolution," *IEEE Transactions on Image Processing*, vol. 23, no. 3, pp. 1340–1351, 2014.

[74] K. Suzuki, I. Horiba, N. Sugie, and M. Nanki, "Extraction of left ventricular contours from left ventriculograms by means of a neural edge detector," *IEEE Transactions on Medical Imaging*, vol. 23, no. 3, pp. 330–339, 2004.

[75] A. Goshtasby and D. A. Turner, "Segmentation of cardiac cine mr images for extraction of right and left ventricular chambers," *IEEE transactions on medical imaging*, vol. 14, no. 1, pp. 56–64, 1995.

[76] J. R. Munkres, *Topology*. Prentice Hall, 2000.

[77] A. Idris, T. Ahmad, and N. Maan, "Homeomorphism between sphere and cube," *Malaysian Journal of Fundamental and Applied Sciences*, vol. 4, no. 2, 2014.

[78] A. Hatcher, "Algebraic topology. 2002," *Cambridge UP, Cambridge*, vol. 606, no. 9.

[79] V. A. Kovalevsky, "Finite topology as applied to image analysis," *Computer vision, graphics, and image processing*, vol. 46, no. 2, pp. 141–161, 1989.

[80] J. Chaussard and M. Couprie, "Surface thinning in 3d cubical complexes," in *Combinatorial Image Analysis*. Springer, 2009, pp. 135–148.

[81] K. Siddiqi and S. Pizer, *Medial representations: mathematics, algorithms and applications*. Springer Science & Business Media, 2008, vol. 37.

[82] B. Raynal and M. Couprie, "Isthmus-based 6-directional parallel thinning algorithms," in *International Conference on Discrete Geometry for Computer Imagery*. Springer, 2011, pp. 175–186.

[83] G. Németh and K. Palágyi, "2d parallel thinning algorithms based on isthmus-preservation," in *Image and Signal Processing and Analysis (ISPA), 2011 7th International Symposium on*. IEEE, 2011, pp. 585–590.

[84] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational homology*. Springer Science & Business Media, 2006, vol. 157.

[85] M. Spivak, *Calculus on manifolds: a modern approach to classical theorems of advanced calculus*. Westview Press, 1971.

[86] J. Milnor, *Morse Theory.(AM-51)*. Princeton university press, 2016, vol. 51.

[87] D. Eberly, R. Gardner, B. Morse, S. Pizer, and C. Scharlach, "Ridges for image analysis," *Journal of Mathematical Imaging and Vision*, vol. 4, no. 4, pp. 353–373, 1994.

[88] A. J. Zomorodian, *Topology for computing*. Cambridge university press, 2009, vol. 16.

[89] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical morse complexes for piecewise linear 2-manifolds," in *Proceedings of the seventeenth annual symposium on Computational geometry*. ACM, 2001, pp. 70–79.

[90] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Morse-smale complexes for piecewise linear 3-manifolds," in *Proceedings of the nineteenth annual symposium on Computational geometry*. ACM, 2003, pp. 361–370.

[91] M. G. Crandall and P.-L. Lions, "Viscosity solutions of hamilton-jacobi equations," *Transactions of the American Mathematical Society*, vol. 277, no. 1, pp. 1–42, 1983.

[92] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 117–156, 1998.

[93] M. Kolomenkin, I. Shimshoni, and A. Tal, "Multi-scale curve detection on surfaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 225–232.

[94] M. Basseville, I. V. Nikiforov *et al.*, *Detection of abrupt changes: theory and application*. Prentice Hall Englewood Cliffs, 1993, vol. 104.

[95] H. V. Poor and O. Hadjiliadis, *Quickest detection*. Cambridge University Press Cambridge, 2009, vol. 40.

[96] V. V. Veeravalli and T. Banerjee, "Quickest change detection," *Academic press library in signal processing: Array and statistical signal processing*, vol. 3, pp. 209–256, 2013.

[97] G. D. Hugo, E. Weiss, W. C. B. Sleeman, K. Salim, L. Paul J., Jun, and J. F. Williamson, "Data from 4d lung imaging of nsclc patients," The Cancer Imaging Archive, http://doi.org/10.7937/K9/TCIA.2016.ELN8YGLE.

[98] B. Lassen, E. M. van Rikxoort, M. Schmidt, S. Kerkstra, B. van Ginneken, and J.-M. Kuhnigk, "Automatic segmentation of the pulmonary lobes from chest ct scans based on fissures, vessels, and bronchi," *IEEE transactions on medical imaging*, vol. 32, no. 2, pp. 210–222, 2013.

[99] C. Xiao, B. C. Stoel, M. E. Bakker, Y. Peng, J. Stolk, and M. Staring, "Pulmonary fissure detection in ct images using a derivative of stick filter," *IEEE transactions on medical imaging*, vol. 35, no. 6, pp. 1488–1500, 2016.

[100] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 519–528.

[101] T. Brunsch, K. Cornelissen, B. Manthey, H. Roglin, and C. Rosner, "Smoothed analysis of the successive shortest path algorithm," *SIAM Journal on Computing*, vol. 44, no. 6, pp. 1798–1819, 2015.

[102] L. R. Ford Jr and D. R. Fulkerson, *Flows in networks*. Princeton university press, 2015.

[103] B. Dezső, A. Jüttner, and P. Kovács, "Lemon–an open source c++ graph template library," *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 23–45, 2011.

[104] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[105] R. Wiemker, T. Bülow, and T. Blaffert, "Unsupervised extraction of the pulmonary interlobar fissures from high resolution thoracic ct data," in *International Congress Series*, vol. 1281. Elsevier, 2005, pp. 1121–1126.

[106] E. M. van Rikxoort, B. van Ginneken, M. Klik, and M. Prokop, "Supervised enhancement filters: Application to fissure detection in chest ct scans," *IEEE Transactions on Medical Imaging*, vol. 27, no. 1, pp. 1–10, 2008.

A Papers Published and Under Review

• Marei Algarni and Ganesh Sundaramoorthi, "SurfCut: Surfaces of Minimal Paths From Topological Structures", *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2017 (Submitted)

• Marei Algarni and Ganesh Sundaramoorthi, "SurfCut: Free-Boundary Surface Extraction", *European Conference on Computer Vision*, 2016 (Accepted)

• Naeemullah Khan, Marei Algarni, Anthony Yezzi, Ganesh Sundaramoorthi, "Shape-tailored local descriptors and their application to segmentation and tracking", *IEEE Conference on Computer Vision and Pattern Recognition*, 2015 (Accepted)

• Marei Algarni and Ganesh Sundaramoorthi, "System and Methods for Free-Boundary Surface Extraction", *U.S. Patent Pending*, April 2016