Cell Path Reconstruction Using 3D Digital Inpainting

by

Anthony Schmieder

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2013

Abstract

Digital inpainting is the reconstruction of a missing or damaged region in a digital image. Intensity values in the missing region are approximated using information near the boundary of the region. Some applications include repair of chipped paintings, repair of rips in paper photographs, and removal of unwanted objects from photographs. In this thesis, we review 2D digital inpainting techniques, examine the application of 3D digital inpainting to cell path reconstruction, and propose a new inpainting technique inspired by the cell path reconstruction problem.

Cell path reconstruction is the estimation of the shape and position of living cells in videos recorded using fluorescence microscopy. This procedure is necessary because in a particular phase of the life cycle of some cells, fluorescent light passes through the cells with an undetectable change in wavelength and they vanish from the frame. This leads to misleading results when, for example, the number of cells in a particular frame is counted. We transform the position/shape estimation problem into a 3D shape reconstruction problem by stacking the frames of the video to form a 3D volume. In this volume, cell paths form tubes with missing segments where cells have vanished. We apply elastica inpainting to the 3D tube reconstruction problem and introduce a new 3D inpainting model to overcome difficulties with a direct generalization to 3D of 2D elastica.

Acknowledgements

We gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada and the Ontario Ministry of Training, Colleges, and Universities for their financial support of this work.

Dedication

Dedicated to Charlotte Jean Schmieder.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Fluorescence microscopy is a tool for magnification and imaging. A specimen is said to *fluoresce* if it absorbs and radiates fluorescent light almost immediately. The radiated light typically has a longer wavelength than the absorbed light. If a specimen is irradiated with fluorescent light of a known wavelength and a fluorescent light camera is filtered to only detect that wavelength, the specimen will be visible in the captured image. Fluorescence microscopy is useful in the study of living cells because fluorescent light is not toxic to the cells. Unfortunately, in a particular phase of the life cycle of some cells, fluorescent light passes through them with a change in wavelength that is too small to detect; they no longer fluoresce. This phenomenon is known as *fading* [43] and is illustrated in Figure 1.1.

Fading causes cells to disappear temporarily in videos of fluorescence microscopy experiments. This leads to confusing results if the purpose of an experiment is, for example, to count the number of cells in each frame of the video. It is therefore desirable to estimate the shape and position of cells in frames where they have vanished. We approach this problem by transforming it to an image reconstruction problem in 3D.

First, a 3D volume is formed by stacking the frames of the video. The path of the cell forms a *tube* in this volume as illustrated in Figure 1.2. In this 3D image, fading will appear as a break in the tube where the cell has vanished. The problem is now to reconstruct missing data in a digital image of a 3D tube. A technique called digital inpainting has been

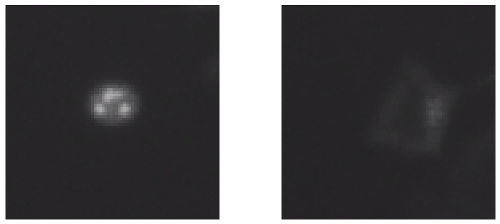

(a) Fluorescent nucleus (b) Faded nucleus

Figure 1.1: Fading

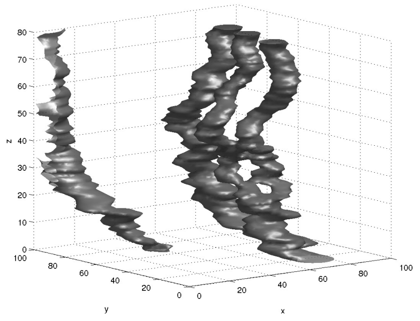


Figure 1.2: Cell path form tubes in stacked images

applied in 2D image processing to estimate missing regions in images [38]. We generalize a 2D inpainting method to 3D and use it to reconstruct broken cell paths in the 3D volume.

This idea is closely related to the arteriosclerosis quantification measure of Dong et al. [21]. In their work, the volume of plaque in a blood vessel is measured by computing the interior volume of an affected region of the vessel then removing that region and recovering an estimate using 3D inpainting. The volume of the estimate is computed and the difference in volumes is used to estimate the volume of the plaque. However, their method only considers the reconstruction of surfaces in 3D. A method for cell path inpainting should also reconstruct the interior of cells.

In this thesis, we first define the terms and notation that we use to discuss digital inpainting models. We then explain the necessary mathematical background including numerical methods for solving the inpainting models and the criteria and test cases we use to select a 2D method for generalization to 3D. Next, we evaluate methods for 2D digital inpainting and select elastica inpainting for cell path reconstruct. We consider the numerical solution of the 2D elastica model in detail. Finally, we discuss the generalization to 3D of the 2D elastica model and its application to cell path reconstruction. We find that a direct generalization to 3D of the existing 2D elastica model yields unsatisfactory results, so we propose a new 3D inpainting technique inspired by cell path reconstruction.

Chapter 2

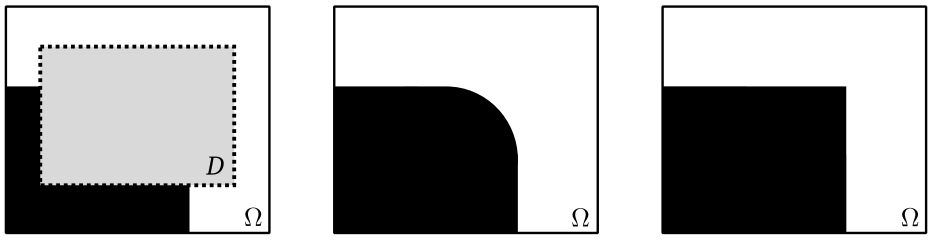
Mathematical Background

In this section, we first present the notation and mathematical tools that we use to phrase and solve inpainting problems. We then explain the criteria that we use to select a 2D inapinting technique for cell path reconstruction in 3D.

2.1 Definitions

An *image* is a map $u : \Omega \to [0, 1]$ where the image domain $\Omega \subset \mathbb{R}^2$ is assumed to be rectangular. All images are assumed to be grayscale with intensity values in $[0, 1]$: 0=black, 1=white. When a method uses derivatives of u, we assume that u is sufficiently smooth.

For image inpainting, an *inpaint domain* $D \subset \Omega$ and an initial image u^0 are given. The boundary of D is defined as $\operatorname{closure} D \setminus \operatorname{interior} D$ and is denoted ∂D. u^0 is assumed to be unknown on D. The problem is to find an image u such that $u \approx u^0$ on $\Omega \setminus D$ and u on D is constructed to look reasonable to a human observer. The definition of reasonable depends on the problem and there are typically multiple reasonable solutions. Figure 2.1 illustrates two reasonable solutions for the same inpainting problem. Inpainting methods are distinguished by which reasonable solution they select for a particular problem. For 3D cell path inpainting, we require a method that recovers smooth tubes of near-constant radius across large inpaint domains.



(a) Problem (b) Solution 1 (c) Solution 2

Figure 2.1: An inpainting problem and two reasonable solutions

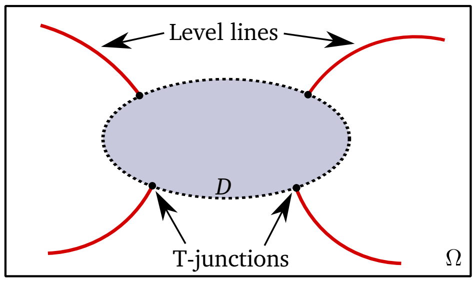
In our discussion, we use the concepts of level lines, T-junctions, the gradient vector, and the unit normal vector. A *level line* is a line in an image along which the image intensity is constant. A *T-junction* is a point where a level line intersects ∂D. Level lines and T-junctions are illustrated in Figure 2.2. The gradient ∇u of an image $u = u(x_1, x_2)$ is

$$\nabla u = \left[\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}\right]^\top.$$

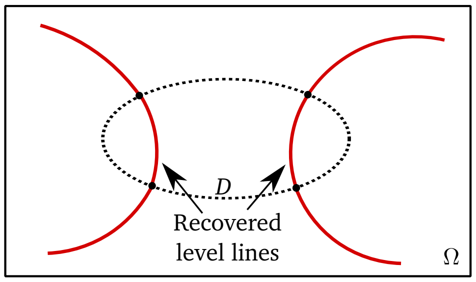
Figure 2.3 shows a 2D image u and the graph of u depicted as a 3D surface. The locations of some level lines are shown as curves in the $u = 0$ plane of the 3D plot. The arrows in the same illustration are ∇u evaluated at different points in the image domain Ω. Note that $|\nabla u|$ is large at points where the surface u has steep incline and small where it has shallow incline. The gradient at any point in the image is normal to the level line passing through that point [26, p. 291], so the *unit normal vector* for a level line may be computed as

$$\vec{n} = \frac{\nabla u}{|\nabla u|}.$$

Our notation and definitions follow those of Aubert and Kornprobst [2].

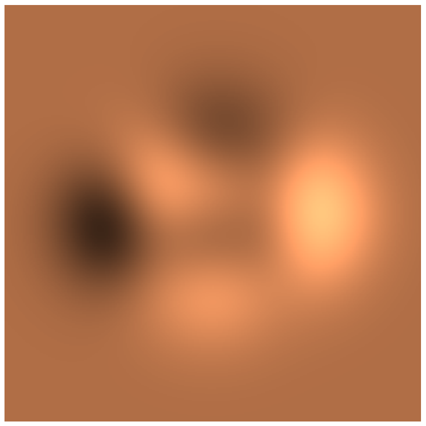


(a) An inpainting problem

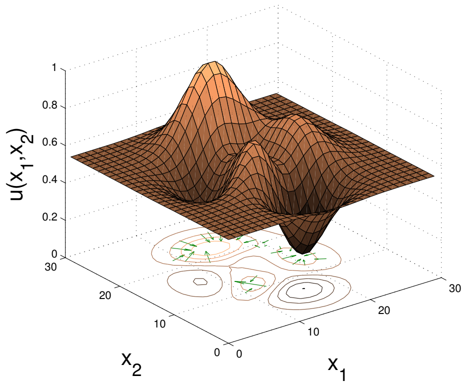


(b) A reasonable solution

Figure 2.2: Illustration of the inpainting problem



(a) An image u



(b) Some level lines and gradient vectors for u

Figure 2.3: Level lines and gradient vectors

2.2 Solution Techniques

The 2D geometric inpainting models considered in this work are formulated as the minimization:

$$\underset{u}{\operatorname{argmin}}\, J(u) \tag{2.1}$$

where

$$J(u) = \int_{\Omega} f(x, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) dxdy.$$

Depending on the model, the functional J may be non-linear and non-convex. This gives rise to difficult minimization problems. In this section, we present several techniques that have been applied to these problems.

2.2.1 Calculus of Variations

A useful tool for solving minimization models in numerical image processing is the calculus of variations. We give a brief explanation of how the subject relates to the geometric models described in Section 3.1.

The calculus of variations was developed to analyze minimization problems of the form

$$\underset{u}{\operatorname{argmin}} \int_{x_1}^{x_2} f(x, u(x), u'(x)) dx \tag{2.2}$$

subject to $u(x_1) = u_1$ and $u(x_2) = u_2$ where f and u are twice differentiable. It can be shown [46] that any u that minimizes (2.2) and satisfies the given conditions is also a solution to the partial differential equation

$$\frac{\partial f}{\partial u} - \frac{d}{dx}\left(\frac{\partial f}{\partial u'}\right) = 0. \tag{2.3}$$

Equation (2.3) is known as the *Euler-Lagrange equation.* This result may be generalized to functions u with multidimensional domains and functions f of higher order derivatives of u [46]. For example, for a given domain $\Omega \subset \mathbb{R}^2$, a function $u : \Omega \to \mathbb{R}$, and the minimization problem

$$\operatorname*{argmin}_u \int_\Omega f(x, y, u_x, u_y, u_{xx}, u_{xy}, u_{yy}) dxdy \tag{2.4}$$

where u is given on $\partial\Omega$, a minimizer u of (2.4) must satisfy

$$\frac{\partial f}{\partial u} - \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial u_x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial u_y}\right) + \frac{\partial^2}{\partial x^2}\left(\frac{\partial f}{\partial u_{xx}}\right) + \frac{\partial^2}{\partial x \partial y}\left(\frac{\partial f}{\partial u_{xy}}\right) + \frac{\partial^2}{\partial y^2}\left(\frac{\partial f}{\partial u_{yy}}\right) = 0. \tag{2.5}$$

When Equation (2.4) is compared to the total variation and elastica inpainting models given in Equations 3.4 and 3.10, the utility of the calculus of variations in numerical image processing becomes apparent. To solve minimizations over a 3D image domain as in the models of Chapter 4, we use the following formula for the Euler-Lagrange equation:

$$\begin{aligned} &\frac{\partial f}{\partial u} - \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial u_x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial u_y}\right) - \frac{\partial}{\partial z}\left(\frac{\partial f}{\partial u_z}\right) \\ &\quad + \frac{\partial^2}{\partial x^2}\left(\frac{\partial f}{\partial u_{xx}}\right) + \frac{\partial^2}{\partial y^2}\left(\frac{\partial f}{\partial u_{yy}}\right) + \frac{\partial^2}{\partial z^2}\left(\frac{\partial f}{\partial u_{zz}}\right) \\ &\quad + \frac{\partial^2}{\partial x \partial y}\left(\frac{\partial f}{\partial u_{xy}}\right) + \frac{\partial^2}{\partial x \partial z}\left(\frac{\partial f}{\partial u_{xz}}\right) + \frac{\partial^2}{\partial y \partial z}\left(\frac{\partial f}{\partial u_{yz}}\right) = 0. \end{aligned} \tag{2.6}$$

2.2.2 Solution of the Euler-Lagrange Equation

In numerical image processing, two techniques are commonly used to solve the Euler-Lagrange equation: artificial time marching and fixed point iteration [16]. For example, consider the total variation (TV) minimization problem:

$$\operatorname*{argmin}_u \int_\Omega |\nabla u| dxdy + \frac{\lambda}{2} \int_{\Omega \setminus D} (u^0 - u)^2 dxdy \tag{2.7}$$

where λ is a constant. This model is discussed in Section 3.1.2, but here we only consider the minimization. Using the formula given in Equation (2.5), the Euler-Lagrange equation for (2.7) is:

$$-\nabla \cdot \frac{\nabla u}{|\nabla u|} + \lambda_{\Omega\backslash D} \cdot (u - u^0) = 0 \tag{2.8}$$

where

$$\lambda_{\Omega\backslash D}(x) = \begin{cases} \lambda, & \text{if } x \in \Omega\backslash D \\ 0, & \text{otherwise} \end{cases}.$$

This expression is equivalent to the gradient of (2.7). Therefore, the minimization may be solved using a steepest descent method by introducing the artificial time variable t. This gives

$$\frac{\partial u}{\partial t} = \nabla \cdot \frac{\nabla u}{|\nabla u|} + \lambda_{\Omega\backslash D} \cdot (u^0 - u) \tag{2.9}$$

When this equation is solved using an explicit numerical scheme, the time step corresponds to the step length in the gradient descent [16]. When faster convergence is required, a fixed point iteration may be used to solve (2.8). A sparse linear system is obtained by lagging the nonlinear term in the diffusion coefficient [16]:

$$-\nabla \cdot \frac{\nabla u^{n+1}}{|\nabla u^n|} + \lambda_{\Omega\backslash D} \cdot (u^{n+1} - u^0) = 0. \tag{2.10}$$

An initial guess is chosen for u^0 and the linear system is solved repeatedly until convergence. These techniques work well for the total variation model, but for higher-order nonlinear models such as elastica, the time step in artificial time marching must be impractically small and it is difficult to construct a stable fixed point method. We consider these issues in detail in Section 3.1.4.

2.2.3 Gradient Flow Methods

Gradient flow methods [23] provide fast, unconditionally stable techniques for solving gradient systems.

Definition 1. *A* gradient system *is an initial value system of the form*

$$\begin{cases} \frac{\partial u}{\partial t} = -\nabla f(u) \\ u(0) = u^0 \end{cases} \tag{2.11}$$

for $f : \mathbb{R}^d \to \mathbb{R}$ satisfying

$$\begin{cases} f(u) \geq 0 & \forall u \in \mathbb{R}^d \\ f(u) \to \infty & \text{as } |u| \to \infty \\ \langle H(f)(u)u, u\rangle \geq \lambda & \forall u \in \mathbb{R}^d \end{cases}$$

where $\lambda \in \mathbb{R}$, $H(f)$ is the Hessian matrix of f, and $\langle \cdot, \cdot \rangle$ is the inner product.

Eyre [23] solves such a system by splitting f into

$$f(u) = f_c(u) - f_e(u) \tag{2.12}$$

where f_c and f_e are strictly convex for all u. Then the gradient system becomes

$$\frac{\partial u}{\partial t} = \underbrace{-\nabla f_c}_{\substack{\text{steepest}\\ \text{descent}}} \underbrace{+\nabla f_e}_{\substack{\text{hill}\\ \text{climbing}}} .$$

It is solved semi-implicitly using the numerical scheme:

$$u^{n+1} = u^n + \Delta t \left(\nabla f_e(u^n) - \nabla f_c(u^{n+1})\right)$$

where Δt is the time step. Eyre [23] demonstrates the unconditional gradient stability of this scheme (i.e. $f(u^{n+1}) \leq f(u^n)$). Unconditional stability is desirable for high-order inpainting methods, but the models discussed in this thesis are not immediately in the form of Equation (2.11). Furthermore, although the decomposition in Equation (2.12) is guaranteed to exist, it can be difficult to compute [50, 23]. In the next section, we justify the application of gradient flow splitting methods to digital inpainting models by demonstrating that the Euler-Lagrange equation is the gradient of the energy functional in the associated minimization problem.

2.2.4 The Euler-Lagrange Equation as a Gradient Flow

Consider again the minimization from Section 2.2.1:

$$\underset{u}{\operatorname{argmin}}\, J(u) \tag{2.13}$$

where

$$J(u) = \int_{x_1}^{x_2} f(x, u, u')dx$$

subject to $u(x_1) = u_1$ and $u(x_2) = u_2$. We derive the Euler-Lagrange equation for this minimization following [46]. We then consider the definition of the gradient to demonstrate that the Euler-Lagrange equation is the gradient for the functional J. Finally, we obtain a gradient flow by applying artificial time marching to the Euler-Lagrange equation.

To derive the Euler-Lagrange equation, we assume the existence of a twice differentiable function that satisfies the minimization in Equation (2.13). We then derive a differential equation that must be satisfied by such a function. Suppose $u = u(x)$ is a minimizer for Equation (2.13) and consider a perturbation of $u(x)$:

$$U(x) = u(x) + \epsilon\eta(x)$$

where $\eta(x)$ is continuous and differentiable, $\eta(x_1) = \eta(x_2) = 0$, and $\epsilon \in \mathbb{R}$. Then the integral

$$I(\epsilon) = \int_{x_1}^{x_2} f(x, U(x), U'(x))dx$$

has a minimum at $\epsilon = 0$, so $I'(0) = 0$. By differentiating I with respect to ϵ and integrating by parts [46, p. 22], we obtain

$$I'(0) = \int_{x_1}^{x_2} \left(\frac{\partial f}{\partial u} - \frac{d}{dx}\left(\frac{\partial f}{\partial u'} \right) \right) \eta dx = 0. \tag{2.14}$$

Since this equality holds for arbitrary η, the fundamental lemma of the calculus of variations [46] implies that

$$\frac{\partial f}{\partial u} - \frac{d}{dx}\left(\frac{\partial f}{\partial u'}\right) = 0.$$

This expression is the Euler-Lagrange equation. To demonstrate that it is the gradient of J, we consider the definition of the gradient.

The gradient of a functional J at the point u is the unique vector $\nabla J(u)$ such that

$$DJ(u)[\eta] = \langle \nabla J(u), \eta \rangle \tag{2.15}$$

for all vectors η where $DJ(u)[\eta]$ is the directional derivative of J in the direction of η at the point u and $\langle v, w\rangle = \int vw dx$ is the inner product [26]. The integral $I'(0)$ computed in Equation (2.14) is also the directional derivative of J. By comparing that result with the definition of the gradient in Equation (2.15), we see that the Euler-Lagrange equation is the gradient for J:

$$\begin{aligned} DJ(u)[\eta] &= \frac{d}{d\epsilon} J(u+\epsilon\eta)\bigg|_{\epsilon=0} \\ &= \int_{x_1}^{x_2} \left(\frac{\partial f}{\partial u} - \frac{d}{dx}\left(\frac{\partial f}{\partial u'}\right)\right) \eta dx \\ &= \left\langle \frac{\partial f}{\partial u} - \frac{d}{dx}\left(\frac{\partial f}{\partial u'}\right), \eta \right\rangle. \end{aligned}$$

So the minimization in Equation (2.13) may be written as a gradient flow by applying artificial time marching to the Euler-Lagrange equation:

$$\frac{\partial u}{\partial t} = -\nabla_f J(u).$$

We write ∇_f to distinguish the gradient of the functional J from the gradient ∇u of the function u.

Taylor and Cahn [44] apply this approach to the Cahn-Hilliard equation. Bertozzi et al. [7, 6] apply the Cahn-Hilliard equation to inpainting of binary images and provide a detailed analysis of the stability of the numerical scheme and its performance as an inpainting method. Schönlieb and Bertozzi [42] use gradient flow methods to derive new higher-order inpainting models. Finally, Brito-Loeza and Chen [10] apply the gradient flow idea to the elastica inpainting model. They propose several splittings of the Euler-Lagrange equation and fast numerical techniques for solving them. We apply their USTM2 technique to solve the mean/Gaussian 3D inpainting model in Section 4.4.

2.2.5 Direct Minimization of the Energy Functional

The numerical results reported later in this thesis demonstrate that the Euler-Lagrange PDE can be difficult to solve numerically. Therefore, we also consider the direct application of numerical optimization techniques to Equation (2.1). Given an *objective function* $J : \mathbb{R}^n \to \mathbb{R}$ and a set of constraint functions $c_i : \mathbb{R}^n \to \mathbb{R}; i \in \mathcal{E} \cup \mathcal{I}$ where $\mathcal{E}$ and $\mathcal{I}$ are disjoint finite sets of indices, the *constrained optimization problem* can be written as

$$\underset{u \in \mathbb{R}^n}{\operatorname{argmin}} J(u) \qquad \text{subject to} \begin{cases} c_i(u) = 0, & i \in \mathcal{E} \\ c_i(u) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{2.16}$$

The function $c_i; i \in \mathcal{E}$ are called the *equality constraints* and the functions $c_i; i \in \mathcal{I}$ are called the *inequality constraints.* A point $u \in \mathbb{R}^n$ is called *feasible* if all constraints c_i in Equation (2.16) are satisfied. A point u^* is called a *local solution* of Equation (2.16) if u^* is feasible and there is a neighbourhood $\mathcal{N}$ of u^* such that $J(u^*) \leq J(u)$ for all feasible $u \in \mathcal{N}$ [37]. The numerical optimization techniques considered in this thesis search for a local solution to Equation (2.16) by iteratively improving an initial guess for u until a local solution is reached.

There are two classes of numerical optimization techniques: *line search methods* and *trust region methods* [37]. Given a current iterate u^k at iteration k, a line search method selects a feasible descent direction $p^k \in \mathbb{R}^n$ and a step length $\alpha^k \in \mathbb{R}$ to approximately minimize J along the direction p^k:

$$\alpha^k = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} J(u^k + \alpha p^k).$$

Then u is updated as

$$u^{k+1} = u^k + \alpha^k p^k.$$

If the minimization problem is unconstrained, then a trust region method may be applied. A trust region method also solves a minimization subproblem at each iteration, but its search is not restricted to a line. At iteration k, it constructs a quadratic approximation m^k of the objective J such that $m^k(p) \approx J(u^k + p)$. The approximation is assumed to be accurate within some radius Δ^k of the current iterate u^k and the subproblem is solved inside that radius. Then the step p^k is computed by solving

$$p^k = \operatorname*{argmin}_{p \in \mathbb{R}^n} m^k(p) \qquad \text{subject to } \|p\| \leq \Delta^k.$$

Numerical optimization techniques differ in their implementations of the details of the line search and trust region approaches. The choice of a step vector p^k for line search methods and the approximation m^k for trust region techniques are two examples. Specialized techniques exist for different classes of objective and constraint functions. We apply numerical optimization to the 2D and 3D elastica inpainting models. These problems are nonlinear, large, and sparse. For such problems, suitable techniques include the log-barrier interior point method [11, 12, 45], sequential quadratic programming [24, 27, 39], and reflective trust region [13, 9]. We experiment with each of these methods.

2.2.6 Dynamic Programming

Masnou [32, 33] solves the elastica inpainting model by detecting T-junctions on the boundary of the inpaint domain and using a dynamic programming approach to match T-junctions that are compatible for reconnection. A pair of T-junctions is *compatible* if both T-junctions are at the same level (i.e. same colour) and they may be reconnected without intersecting any line connecting two other T-junctions. See Figure 2.4 for an illustration of two sets of compatible T-junction pairs. The optimal set of compatible T-junctions minimizes elastica energy along reconnected level lines. This set is found using

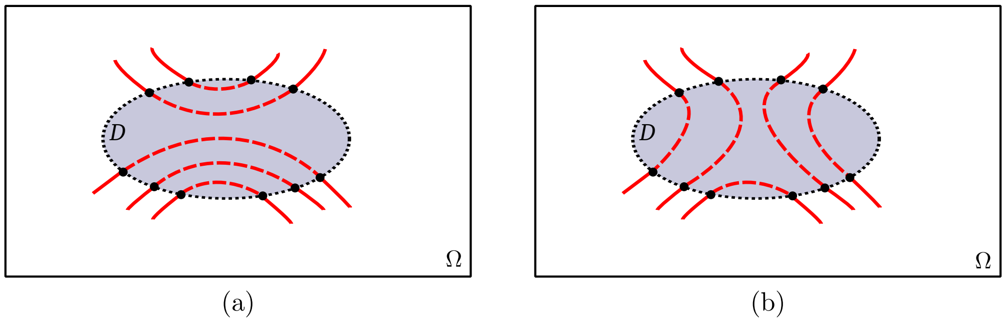


Figure 2.4: Two sets of compatible T-junction pairs

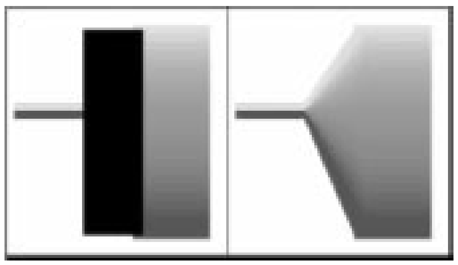


Figure 2.5: Masnou's dynamic programming result (taken from [32])

a dynamic programming approach. Finally, level lines are reconnected by propagating T-junctions through the inpaint domain along elastica-minimizing curves.

Convergence issues are avoided and execution time of the algorithm depends only on the size and shape of the inpaint domain and the number of T-junctions. Since compatible T-junctions are chosen directly, the method can restore level lines across large inpaint domains. Masnou reports impressive results on such cases. For example, see Figure 2.5 taken directly from [32]. The black region in the left-hand image represents the inpaint domain.

Despite this result, it is difficult to accurately detect level lines and their orientation at the inpaint domain boundary when the image is corrupted by noise. In 3D, the problem becomes detection and matching of level surfaces at the boundary of D. Due to the complexity of algorithmically matching level surfaces, we do not consider the dynamic programming approach in this work.

2.3 Selection Criteria for a Cell Path Inpainting Technique

Before generalizing a 2D inpainting technique to 3D we must first select a suitable 2D technique. We identify two properties that a 2D inpainting method must satisfy to be a suitable candidate for application to cell path reconstruction in 3D:

1. The method must reconnect level lines across a large inpaint domain
2. The method must recover level lines that bend smoothly around corners.

Property 1 is necessary because cells can vanish for many frames. For example, the cells depicted in Figure 1.1 are only about twenty pixels in diameter, but vanish for over thirty frames. Property 2 is necessary to ensure natural shape and movement in the reconstructed videos.

To compare inpainting methods, we apply them to two sets of test images. We use the *black bar* test, illustrated in Figure 2.6, to evaluate the tendency of an inpainting method to reconnect level lines across large inpaint domains. The width of the bar is w and the height of the gap is h. The black bar cases used in our experiments are shown in Figure 2.7 and the $w \times h$ dimensions are listed in Table 2.1. We also apply the tests shown in Figure 2.8 to evaluate the tendency of a method to recover smooth curves around corners and to evaluate how a method performs in two cases where a suitable solution is unclear (the images with a gray background). In the test images, the inpaint domains are filled with random intensity values. With these ideas in mind, we are now ready to introduce the inpainting techniques.

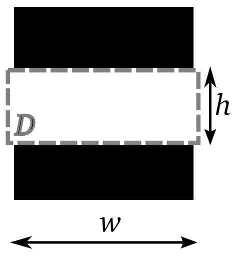


Figure 2.6: The black bar test case

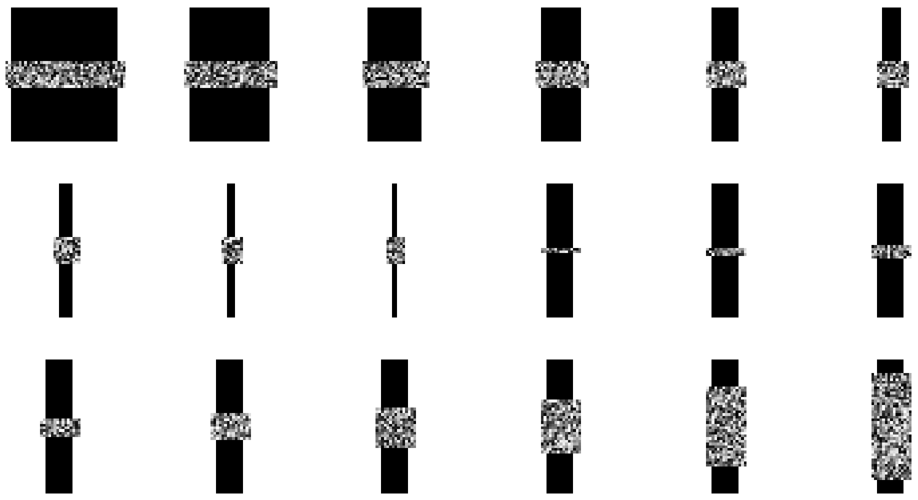


Figure 2.7: Black bar images used in this thesis

w:h ratio	$w \times h$	$w \times h$
4:1	40×10	10×2
3:1	30×10	10×3
2:1	20×10	10×5
1.5:1	15×10	10×7
1:1	10×10	10×10
1:1.5	7×10	10×15
1:2	5×10	10×20
1:3	3×10	10×30
1:4	2×10	10×40

Table 2.1: $w \times h$ dimensions for black bar test cases

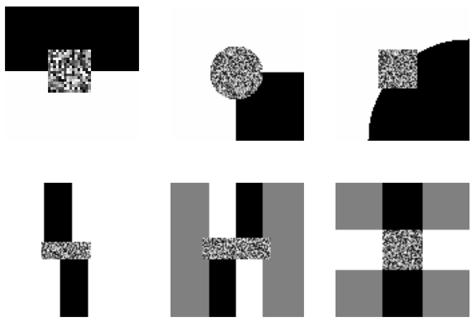


Figure 2.8: Additional test cases

Chapter 3

2D Inpainting Models

We examine three categories of 2D inpainting techniques: geometric, texture synthesis, and exemplar-based. Our 3D technique is geometric, so we examine 2D geometric methods in detail.

3.1 Geometric Methods

In geometric inpainting methods, an image is modelled using ideas from differential geometry. The inpaint domain D is then recovered by optimizing an energy functional that maps the image to a real number. Different image models and different energy functionals produce different inpainting solutions.

3.1.1 Edge Transport Model

Bertalmio, Sapiro, Caselles, and Ballester popularized digital inpainting with the presentation of their paper, "Image Inpainting," at SIGGRAPH in 2000 [4]. They use an intuitive notion of how an artist would repair a damaged painting to construct a partial differential

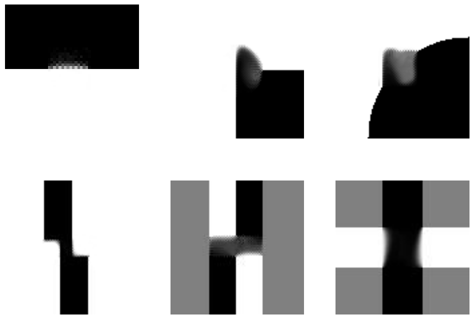


Figure 3.1: Results for edge advection inpainting

equation (PDE) model for inpainting digital images. Their idea is that when an artist repairs a damaged painting, he extends edges into the damaged region on the same trajectory at which they reach the boundary of the region.

To model this idea, Bertalmio et al. first note that since a sharp edge in an image u corresponds approximately to a region with rapidly changing gradient, the edges in u may be located using the Laplacian Δu. The Laplacian will be large on edges of u and small on flat or slowly varying regions of u. Next, they use the advection equation to transport those edges into D parallel to the level lines intersecting ∂D. The 2D advection equation,

$$\frac{\partial u}{\partial t} + \vec{f} \cdot \nabla u = 0,$$

transports the values of u in the direction of the vector field $\vec{f}$. To transport edges in images, Bertalmio et al. replace u by Δu. Since gradient vectors are normal to level lines in u, a $\pi/2$ radian rotation of a vector at a particular point in ∇u yields a vector parallel to the level line passing through the same point. The field of all such vectors is denoted $\nabla^{\perp} u$. This gives the edge transport model:

$$\begin{cases} \frac{\partial u}{\partial t} = \nabla^{\perp} u \cdot \nabla \Delta u & \text{on } D \\ u = u^0 & \text{on } \Omega \backslash D. \end{cases} \tag{3.1}$$

This model can be extended and unified with models of fluid dynamics [3].

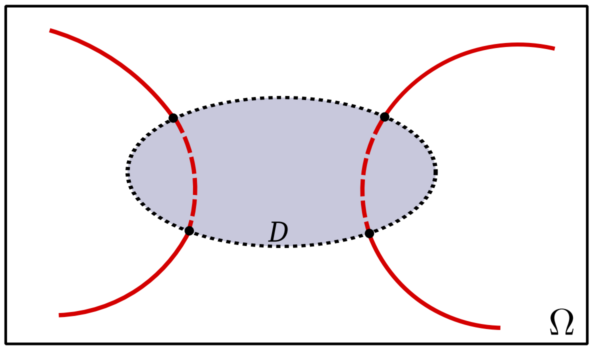


Figure 3.2: Level lines connect to nearby level lines

The edge transport concept is intuitively pleasing, but Figure 3.1 reveals two difficulties with the approach: the model does not match corresponding level lines across D and, since edges are transported in straight lines, the model cannot recover smooth curves. The first difficulty, illustrated in the offset black bar case, is especially troublesome for cell path inpainting. In a 3D model for cell path inpainting, this behaviour would appear as unnatural shape changes and discontinuous movement of cells. In the next section we examine the total variation inpainting model of Chan and Shen [18]. The total variation model solves the problem of matching corresponding level lines across the inpaint domain, but is also unable to recover smooth curves.

3.1.2 Total Variation

The total variation of an image u defined on domain Ω is

$$TV(u) = \int_{\Omega} |\nabla u| dxdy. \tag{3.2}$$

Since it is zero on flat regions and nonzero on nonflat regions, TV can be thought of as a measure of the roughness of u. This idea is applied by Rudin et al. [40] to repair images corrupted by noise. Marquina and Osher [31] extend this idea to simultaneously denoise and deblur an image. Chan and Shen [18] use TV to perform noise removal and inpainting.

To reconnect level lines across D, Chan and Shen use the idea that a level line entering

D at a particular point on ∂D should connect to a nearby point on ∂D at the same level. One implementation of this idea is to reconstruct D using level lines of minimum arc length. This is achieved by choosing values in D that minimize

$$\int_0^1 \left(\int_{u=r} ds \right) dr$$

where to integrate over $u = r$ means to integrate over level lines in the image of intensity r. This integral is difficult to approximate numerically, but the integral over level lines can be transformed to an integral over the area Ω using the coarea formula (see [25] and Theorem 2.5.4 in [2]):

$$\int_0^1 \left(\int_{u=r} ds \right) dr = \int_\Omega |\nabla u| dxdy \tag{3.3}$$

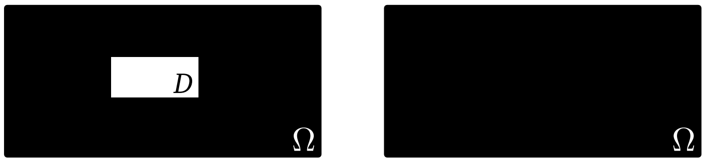
Equation (3.3) reveals that reconstructing D using curves of minimum arc length is equivalent to minimizing $TV(u)$ on D. Adding a fitting constraint to denoise u on $\Omega \backslash D$ gives the TV inpainting model:

$$\underset{u}{\operatorname{argmin}} \int_\Omega |\nabla u| dxdy + \frac{\lambda}{2} \int_{\Omega \backslash D} (u^0 - u)^2 dxdy \tag{3.4}$$

where λ is a constant that allows the user to adjust the amount of smoothing and the level of detail that will remain outside the inpaint domain after smoothing. Note that the TV term is the norm of the gradient ∇u, so minimizing TV across D reconstructs u using a region of zero gradient when possible. This is illustrated in Figure 3.3 where TV is used to inpaint a hole in a constant image. In the input, shown in Figure 3.3(a), $|\nabla u|$ is non-zero only at ∂D, but in the solution, shown in Figure 3.3(b), $|\nabla u|$ is zero on all of Ω.

The results when TV is applied to our set of test cases, shown in Figures 3.4 and 3.5, illustrate that TV inpainting successfully connects corresponding level lines across D. However, these images also illustrate two drawbacks of reconstructing level lines using curves of minimum arc length. First, in an image, curves of minimum arc length are straight lines, so the model can never recover smooth curves.

Second, the TV model does not recover curves across large inpaint domains. This can



(a) An inpainting problem (b) The zero-gradient solution

Figure 3.3: TV inpainting on a constant image with a hole

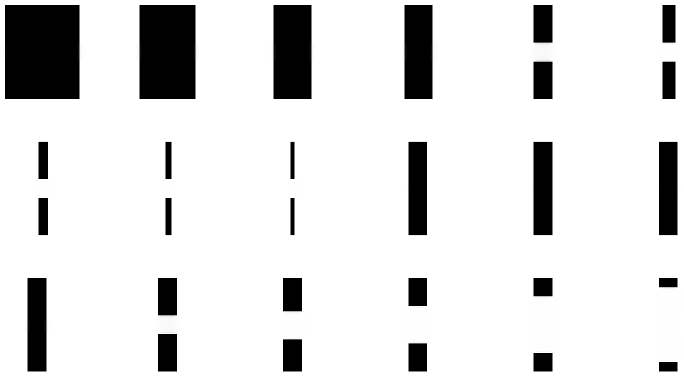


Figure 3.4: Results for TV inpainting on the black bar test cases

be understood by considering the black bar test case shown in Figure 3.6. $TV(u)$ is zero in the constant regions and non-zero along the perimeter of the black bar. If the bar is connected, there will be $2h$ of perimeter in the inpaint domain. If the bar is disconnected, there will be $2w$ of perimeter. Since TV is only non-zero on the boundary of the bar, the model minimizes this perimeter. Therefore, TV inpainting will connect the bar exactly when $h < w$. Gaps in cell paths may be large, so such a limitation is undesirable for a 3D cell path inpainting method. Chan and Shen [15] handle this problem by diffusing level lines having large curvature to extend them into the inpaint domain. This idea is discussed next.

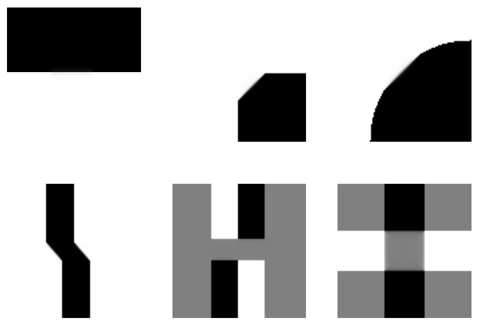
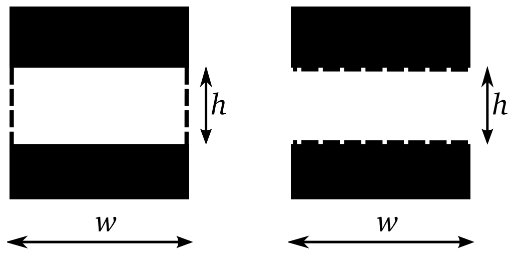


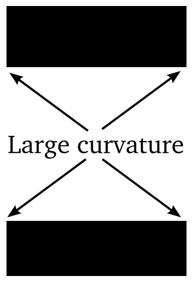
Figure 3.5: Results for TV inpainting on other test cases



(a) Connected: $2h$ of perimeter in D

(b) Disconnected: $2w$ of perimeter in D

Figure 3.6: Reconstruction options for TV inpainting



(a) Large curvature at corners

(b) Zero curvature

Figure 3.7: Curvature penalization in CDD

3.1.3 Curvature Driven Diffusion

In the Curvature Driven Diffusion (CDD) model, Chan and Shen [15] extend the TV model so that level lines having large curvature are penalized in the minimization. The intent is that this will force the model to prefer a closed bar with level lines of zero curvature over a broken bar with large curvature at the corners even when $h > w$ (see Figure 3.7). The curvature of a level line in u may be computed as

$$\kappa = \nabla \cdot \vec{n} = \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right). \tag{3.5}$$

Chan and Shen begin by computing the steepest descent solution of the Euler-Lagrange equation of Equation (3.4) as described in Section 2.2.2:

$$\frac{\partial u}{\partial t} = \nabla \cdot \frac{\nabla u}{|\nabla u|} + (u^0 - u)\lambda_{\Omega \backslash D} \tag{3.6}$$

where

$$\lambda_{\Omega \backslash D}(x) = \begin{cases} \lambda, & \text{if } x \in \Omega \backslash D \\ 0, & \text{otherwise} \end{cases}. \tag{3.7}$$

Next, they note that the first term on the right hand side of Equation (3.6) is the nonlinear diffusion equation with diffusion coefficient $1/|\nabla u|$. Therefore, to solve the TV inpainting model is to find a stationary solution of a diffusion equation where diffusion is diminished by a large gradient. This means that edges with a large gradient are retained by the TV model.

To penalize curvature, Chan and Shen modify the Euler-Lagrange equation by replacing the diffusion coefficient by $|\kappa|^p/|\nabla u|$ when it is computed inside the inpaint domain:

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(\frac{G(\vec{x}, |\kappa|)}{|\nabla u|} \nabla u \right) + (u^0 - u)\lambda_{\Omega \backslash D} \tag{3.8}$$

where

$$G(\vec{x}, s) = \begin{cases} 1, & \vec{x} \in \Omega \backslash D \\ s^p, & \vec{x} \in D, \end{cases}$$

p is chosen to be 1 or 2 depending on the problem, and $\lambda_{\Omega \backslash D}$ is defined as in Equation (3.7). The diffusion coefficient $|\kappa|^p/|\nabla u|$ causes the model to diffuse aggressively in regions where level lines have large curvature while retaining sharp edges in regions where level lines have small curvature.

Chan and Shen's model performs well when reconstructing straight lines. In one difficult back bar test case, the model reconnects the bar when $h = 2w$. However, they note that due to its preference for straight lines, CDD has difficulty recovering smooth curves. Since the tubes formed by cell paths are not straight, CDD is inappropriate for cell path inpainting. Another model, known for its ability to recover smooth curves, is elastica inpainting. We discuss it next.

3.1.4 Elastica

The elastica model can reconstruct smooth curves of nonzero curvature across wide gaps and connect corresponding level lines. It is based on the idea of modelling each level line as a thin, flexible rod and connecting corresponding level lines in such a way that the *bending energy* of the rod is minimized. The bending energy prefers smooth curves and strongly penalizes sharp corners. The bending energy for a curve C is defined as

$$\int_C a + b\kappa^2 ds \tag{3.9}$$

where a and b are constants and κ is curvature. The a term is minimized by a curve of minimum arc length: with the assignments $a = 1$ and $b = 0$, bending energy is reduced to arc length. The b term penalizes curves with high curvature. When the end points are fixed such that the curve cannot be a straight line, the b terms is minimized by smoothing out sharp corners. The bending energy over all level lines in an image is

$$\int_0^1 \left(\int_{u=r} a + b\kappa^2 ds \right) dr$$

where to integrate over $u = r$ means to integrate over all level lines in u that have intensity r. As for TV, the coarea formula [2, 25] may be applied to this expression to transform it into an integral over the image domain Ω:

$$\int_\Omega (a + b\kappa^2)|\nabla u| dxdy$$

where $\kappa(u) = \nabla \cdot \frac{\nabla u}{|\nabla u|}$ is the curvature of the level line in u passing through a particular point in Ω. Finally, by adding the fitting constraint for noise removal outside the inpaint domain, we obtain the elastica model:

$$J_\lambda(u) = \int_\Omega (a + b\kappa^2)|\nabla u| dxdy + \frac{\lambda}{2} \int_{\Omega \backslash D} (u^0 - u)^2 dxdy \tag{3.10}$$

where a, b, and λ are constants and λ adjusts the amount of smoothing as in (3.4). The inpainting problem is then solved by the minimization

$$\underset{u}{\operatorname{argmin}}\, J_\lambda(u).$$

The results in Figures 3.8 and 3.9 demonstrate that the elastica model satisfies the prop-

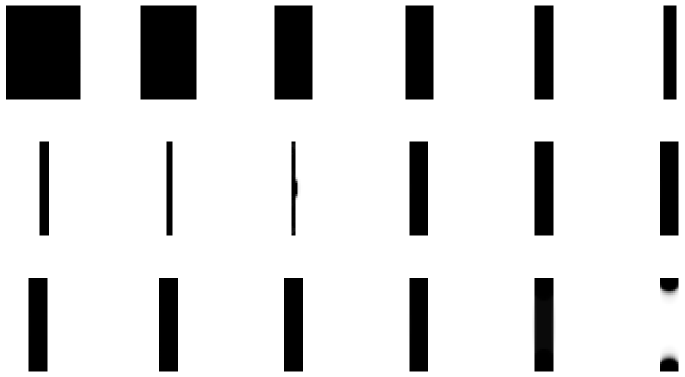


Figure 3.8: Results for elastica inpainting ($a = 0.5, b = 20, \lambda = 100$) on the black bar test cases

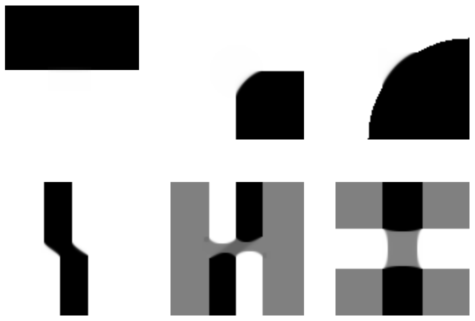


Figure 3.9: Results for elastica inpainting on other test cases

erties given in Section 2.3: it recovers smooth curves and reconnects objects across wide gaps. This suggests that elastica is a promising candidate for 3D cell path reconstruction; we explore this idea in Chapter 4.

The model presented in Equation (3.10) is discussed in detail by Chan, Kang, and Shen [17]. They give a probabilistic motivation for the model and examine other choices for the exponent in the curvature term. Masnou [32] uses a similar model, but does not include the fitting constraint, so his model does not perform denoising. Also, instead of working with the Euler-Lagrange equation, Masnou uses a dynamic programming approach to solve the model (Section 2.2.6). Esedoglu and Shen [22] combine the elastica model with the Mumford-Shah model [36] and form the Mumford-Shah-Euler model. It is high order and difficult to solve, so they approximate it before solving numerically.

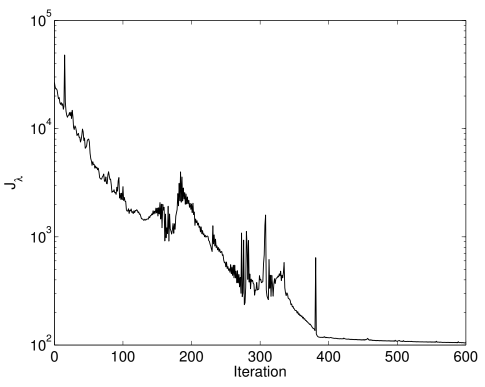
Since we apply elastica inpainting to cell path reconstruction in 3D, we consider in detail the numerical solution on the 2D model.

Numerical Solution of the Euler-Lagrange Equation

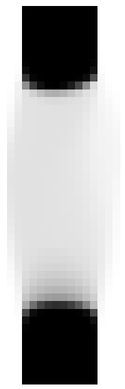
In Figure 3.10, we illustrate the challenging nature of the elastica minimization problem. We present the solution u and a plot of J_λ captured after 600 iterations and after 20,000 iterations of of the fixed point USTM2 scheme of Brito-Loeza and Chen [10]. The input is the black bar test case with $w \times h = 10 \times 30$ and initial data as shown in Figure 2.7. The parameters for the elastica model are $(a, b, \lambda) = (0.5, 20, 100)$ and the stabilization parameter for the numerical scheme is $C_1 = 100$ [10]. Notice the rapid decrease in J_λ over the first 600 iterations as the method progresses quickly toward a disconnected bar. After this, from 600 to 20,000 iterations, J_λ decreases gradually over many iterations as the black bar is slowly recovered. This suggests that the norm of the functional gradient $|\nabla_f J_\lambda|$ is large between the initial value u^0 and the disconnected bar, but small between the disconnected and connected bars.

This is understood by considering the computation of J_λ on the black bar test cases illustrated in Figure 3.11. In these two images, despite the difference in the height of the gap, $\int_\Omega b\kappa^2|\nabla u|dxdy$ is identical. On constant regions, $|\nabla u| = 0$, so the only contributions to $\int_\Omega b\kappa^2|\nabla u|$ come from the level lines in u. Moreover, straight lines have zero curvature, so $\kappa \neq 0$ only on curved level lines. Such curved lines are drawn as solid in Figure 3.11. Since the two figures contain the same curved level lines, $\int_\Omega b\kappa^2|\nabla u|$ is the same in both. Since the a term restricts arc length, this further suggests that for $a \neq 0$ we should find a local minimum at the disconnected bar. We verify this numerically by using a disconnected bar as u^0. The result after 10,000 iterations of elastica with the same parameters as above is shown in Figure 3.12.

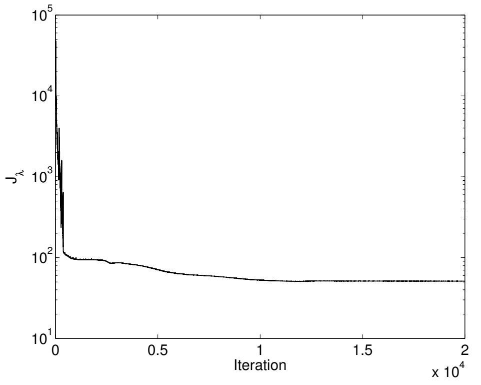
A further complication is that as the bar is recovered we observe severe oscillations in J_λ. This is illustrated in Figure 3.13 where J_λ is plotted over different sequences of 5000 iterations each. We observe in Figures 3.13(b) and 3.13(c) that the general downward



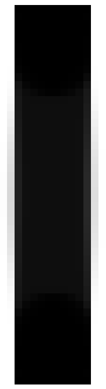
(a) J_λ up to iteration 600



(b) u at iteration 600



(c) J_λ up to iteration 20,000



(d) u at iteration 20,000

Figure 3.10: J_λ and u at different iterations of elastica inpainting

trend in J_λ is dominated by oscillations as the scheme recovers the connected bar. This makes automatic termination difficult and suggests that the numerical scheme is choosing inefficient step directions in the space of u vectors. We can eliminate the oscillations in J_λ by using a numerical optimization technique that guarantees monotonic decrease of J_λ [37]. Such a scheme should have desirable convergence properties and might terminate in fewer iterations. We examine this idea next.

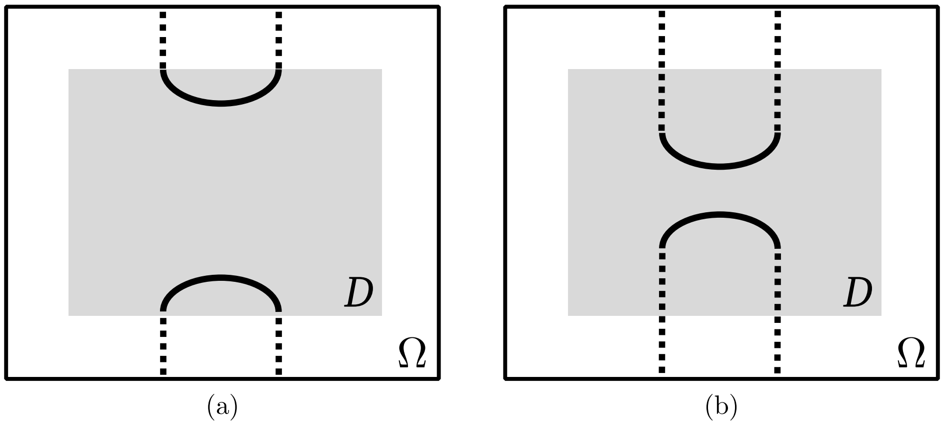


Figure 3.11: Computing J_λ for elastica on the black bar. Dotted lines denote $\kappa = 0$

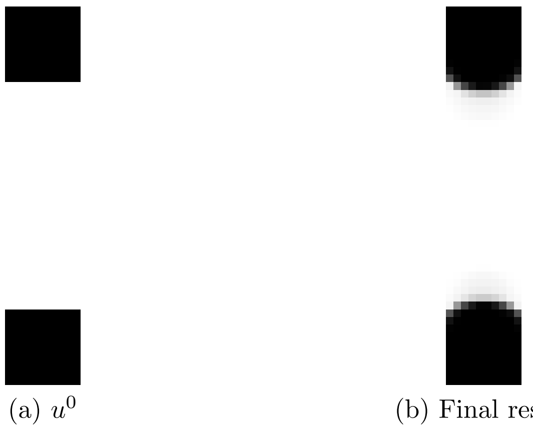
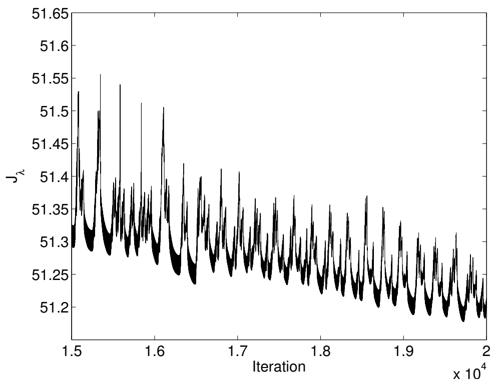
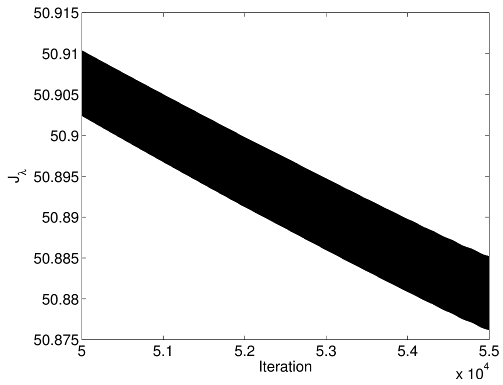


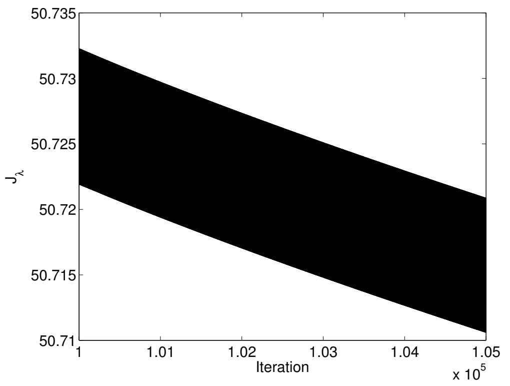
Figure 3.12: Illustration of broken bar local minimum in J_λ



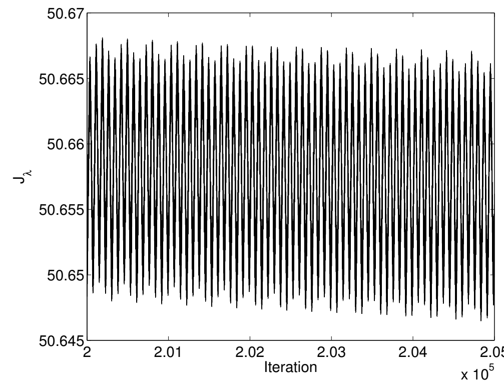
(a) Iterations 15,000–20,000



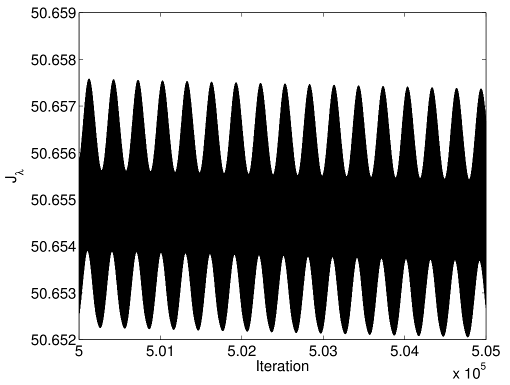
(b) Iterations 50,000–55,000



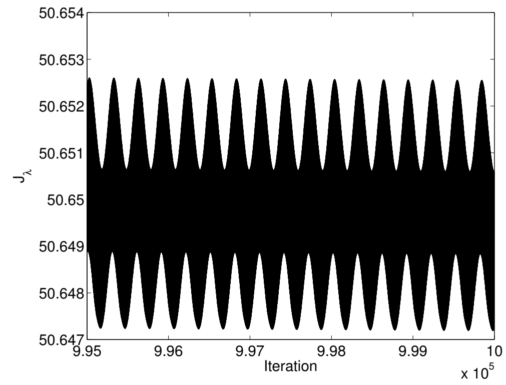
(c) Iterations 100,000–105,000



(d) Iterations 200,000–205,000



(e) Iterations 500,000–505,000



(f) Iterations 995,000–1,000,000

Figure 3.13: Oscillations in J_λ

Solution by Numerical Optimization

In at attempt to obtain faster convergence toward the desired solution, we consider the application of numerical optimization techniques to the elastica minimization. We use the elastica energy functional J_λ from Equation (3.10) directly as the optimization objective. We use the implementations from the optimization toolbox in Matlab R2012a for 64-bit Linux and consider only techniques for which the Matlab implementation is suitable for sparse nonlinear problems. The three techniques that we consider are: log-barrier interior point (LBIP), sequential quadratic programming (SQP), and reflective trust region (RTR) [34, 35].

Configuration of the Matlab Optimization Routines

We impose the box constraint $0 \leq u \leq 1$ on Ω for each of the three optimization techniques. The Matlab implementation of each of the techniques optionally uses the gradient and LBIP and RTR optionally use the Hessian of the discretized objective J_λ. If they are not supplied, then Matlab approximates them using finite differences [34]. We consider an example to demonstrate calculation of the gradient of a discrete objective.

Taking $a = 1, b = 0, \lambda = 0$ reduces the elastica functional to TV:

$$J_\lambda(u) = \int_\Omega |\nabla u| dxdy.$$

Using centered finite differences on a uniform grid with distance Δx between grid points and letting subscripts denote grid coordinates so that $u_{ij} \approx u(i\Delta x, j\Delta x)$ [1], we obtain the discretization

$$\begin{aligned} J_\lambda(u) &\approx \sum_{i,j} |\nabla u_{ij}|_\varepsilon \\ &= \sum_{i,j} \sqrt{\left(\frac{u_{(i+1)j} - u_{(i-1)j}}{2\Delta x}\right)^2 + \left(\frac{u_{i(j+1)} - u_{i(j-1)}}{2\Delta x}\right)^2 + \varepsilon} \end{aligned} \tag{3.11}$$

where ε is introduced to preserve the differentiability of J_λ at $u = 0$ [10]. Then the gradient of the discretized objective is computed by taking a partial derivative with respect to each element u_{ij}. For example, consider the grid illustrated in Figure 3.15. To compute $\frac{\partial J_\lambda}{\partial u_{22}}$, we must consider any terms from the summation (3.11) that depend on u_{22}. Namely $|\nabla u_{21}|_\varepsilon$, $|\nabla u_{12}|_\varepsilon$, $|\nabla u_{32}|_\varepsilon$, and $|\nabla u_{23}|_\varepsilon$. Therefore,

$$\begin{aligned}\frac{\partial J_\lambda}{\partial u_{22}} &= \frac{\partial |\nabla u_{21}|_\varepsilon}{\partial u_{22}} + \frac{\partial |\nabla u_{12}|_\varepsilon}{\partial u_{22}} + \frac{\partial |\nabla u_{32}|_\varepsilon}{\partial u_{22}} + \frac{\partial |\nabla u_{23}|_\varepsilon}{\partial u_{22}} \\ &= \frac{u_{22} - u_{20}}{4\Delta x^2 |\nabla u_{21}|_\varepsilon} + \frac{u_{22} - u_{02}}{4\Delta x^2 |\nabla u_{12}|_\varepsilon} + \frac{u_{22} - u_{42}}{4\Delta x^2 |\nabla u_{32}|_\varepsilon} + \frac{u_{22} - u_{24}}{4\Delta x^2 |\nabla u_{23}|_\varepsilon}.\end{aligned}$$

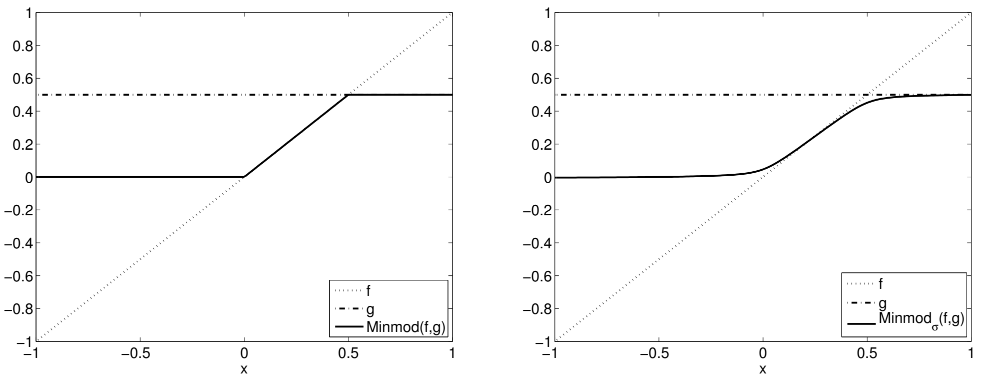
For nonzero b, the gradient is difficult to compute by hand. Instead, we use the `sym` function in the Matlab symbolic toolbox to construct a grid of symbolic variables to represent the grid points of u. We compute the discretized objective over that grid then compute the gradient using the `jacobian` function. The gradient is then evaluated by substituting the symbolic variables for concrete values of a particular u. We export the gradient as C code and compile it using the Matlab compiler tool `mex`. We configure the optimization schemes to approximate the Hessian matrix using BFGS for LBIP and finite differences for RTR.

Additionally, to use a nonzero b, we must discretize the second-order curvature operator κ. We explore two options. First, to ensure a small stencil, we expand the curvature using the divergence form and apply a midpoint discretization [40, 18, 17, 10]:

$$\kappa(u) = \nabla \cdot \frac{\nabla u}{|\nabla u|} = \frac{\partial}{\partial x}\left(\frac{u_x}{|\nabla u|}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{|\nabla u|}\right).$$

Applying the midpoint discretization gives

$$\begin{aligned}\kappa(u_{ij}) \approx \frac{1}{\Delta x^2}\left(\frac{u_{(i+1)j} - u_{ij}}{|\nabla u_{(i+1/2)j}|_\varepsilon}\right) - \frac{1}{\Delta x^2}\left(\frac{u_{ij} - u_{(i-1)j}}{|\nabla u_{(i-1/2)j}|_\varepsilon}\right) \\ + \frac{1}{\Delta x^2}\left(\frac{u_{i(j+1)} - u_{ij}}{|\nabla u_{i(j+1/2)}|_\varepsilon}\right) - \frac{1}{\Delta x^2}\left(\frac{u_{ij} - u_{i(j-1)}}{|\nabla u_{i(j-1/2)}|_\varepsilon}\right)\end{aligned}$$



(a) minmod (b) $\text{minmod}_\sigma, \sigma = 10^{-1}$

Figure 3.14: Comparison of minmod and minmod_σ for $f(x) = x$ and $g(x) = 0.5$

with the norm of a gradient at a midpoint being computed as

$$|\nabla u_{(i+1/2)j}|_\varepsilon = \sqrt{\left(\frac{u_{(i+1)j} - u_{ij}}{\Delta x}\right)^2 + \mu\left(\frac{u_{(i+1)(j+1)} - u_{(i+1)(j-1)}}{2\Delta x}, \frac{u_{i(j+1)} - u_{i(j-1)}}{2\Delta x}\right)^2 + \varepsilon}$$

where μ is the minmod:

$$\mu(x, y) = \text{minmod}(x, y) = \begin{cases} \text{sgn}(x)\min(|x|, |y|), & \text{if } xy > 0 \\ 0, & \text{otherwise.} \end{cases}$$

We compute a differentiable approximation minmod_σ of minmod using the following differentiable approximations:

$$|x|_\sigma = \sqrt{x^2 + \sigma^2}$$
$$\text{sgn}_\sigma(x) = \frac{x}{|x|_\sigma}$$
$$\min_\sigma(x, y) = \frac{1}{2}\left(x + y - |x - y|_\sigma\right).$$

The functions minmod and minmod_σ are compared in Figure 3.14.

A second option to compute a discrete approximation of κ is to fully expand the divergence form and approximate the derivatives using finite differences:

$$\kappa(u) = \frac{\partial}{\partial x}\left(\frac{u_x}{|\nabla u|}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{|\nabla u|}\right) = \frac{u_{xx}u_y^2 - 2u_x u_y u_{xy} + u_{yy}u_x^2}{|\nabla u|^3}.$$

In our numerical experiments, we find that the optimization methods are sensitive to the discretization of κ and the choice of ε. We choose these parameters based on speed of convergence and suitability of the solution for cell path inpainting. If these criteria conflict, we give priority to the suitability of the solution.

We experimentally select stopping tolerances for the numerical schemes by starting with large values and gradually lowering them until the method converges to the desired solution. The parameters for the optimization routines and the elastica model are reported in Table 3.1.

Finally, we alter the default configuration of the LBIP scheme [35]. By default, at each iteration LBIP attempts to compute a direct solution of the KKT equations [37, p. 321]. This method depends on the Hessan and requires it to be positive-definite near the current iterate. Alternatively, LBIP can use a trust region conjugate gradient method [37, p. 101]. For the elastica minimization problem, we find that LBIP converges to a desirable result only when we select conjugate gradient method. The non-convexity and non-linearity of the elastica minimization might lead to inaccuracy of the Hessian approximation and difficulty in the direct KKT solution scheme. We therefore configure Matlab to use the conjugate gradient scheme using `'SubproblemAlgorithm'='cg'` in the `fmincon` options.

Numerical Results

We tested the numerical optimization techniques on the $w \times h = 10 \times 12$ and $w \times h = 10 \times 30$ black bar cases with initial guess 0.5 on D. The black bar is recovered for both cases by all optimization techniques. The plots of J_λ in Figure 3.16 demonstrate the efficiency of the step directions chosen by the numerical optimization schemes.

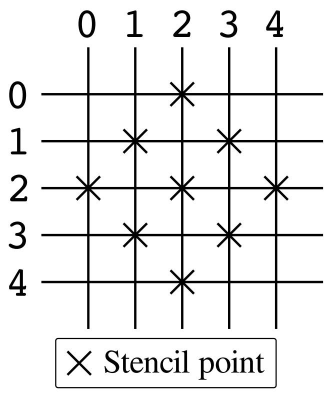


Figure 3.15: Stencils of $|\nabla u_{21}|$, $|\nabla u_{12}|$, $|\nabla u_{32}|$, and $|\nabla u_{23}|$

	LBIP	**SQP**	**RTR**
a	0.5	0.5	0.5
b	20	20	20
Δx	1/13	1/13	1/13
ε	10^{-2}	10^{-1}	10^{-2}
σ	10^{-2}	10^{-2}	10^{-2}
curvature	minmod	fully expanded	minmod
tolx	10^{-4}	10^{-3}	10^{-3}
tolfun	10^{-5}	10^{-2}	10^{-5}
tolcon	10^{-2}	10^{-2}	10^{-2}
maxiter	∞	∞	∞
maxfunevals	∞	∞	∞
hessian	BFGS	n/a	fin-diff-grads

Table 3.1: Parameters for Matlab optimization of 2D elastica

	LBIP	**SQP**	**RTR**
CPU Time (minutes)	4.6	2.9	0.4
Iterations	2834	313	52
# J_λ evaluations	4860	3385	53

Table 3.2: Optimization time comparison for 2D elastica on $w \times h = 10 \times 12$ black bar

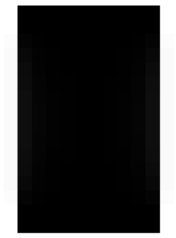
	LBIP	**SQP**	**RTR**	**FP**
CPU Time (minutes)	62.0	761.2	15.2	8.1
Iterations	7367	1028	174	17447
# J_λ evaluations	12314	14610	175	n/a

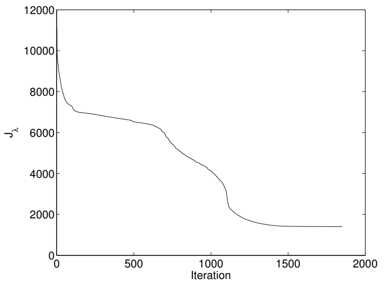
Table 3.3: Optimization time comparison for 2D elastica on $w \times h = 10 \times 30$ black bar

Execution times, iteration counts for the optimization scheme, and number of evaluations of the objective J_λ are reported in Tables 3.2 and 3.3. Table 3.3 includes the results using the gradient flow fixed point method (FP) for elastica [10]. Due to the convergence issues outlined above, convergence of the fixed point method is detected by visual inspection of u. We perform the experiments on a 3 GHz Intel Xeon X5472 with 16 GB of RAM and compute execution time using the Matlab `cputime` function.

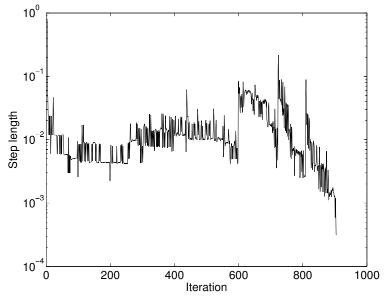
The numerical optimizations routines in Matlab require fewer iterations than the fixed point scheme, but take more time per iteration. However, the CPU execution time of reflective trust region and the fixed point scheme differ only by a factor of two. The additional benefits of monotonically decreasing J_λ and accurate convergence detection are attractive incentives to prefer reflective trust region over the fixed point scheme.

The results in this section demonstrate that elastica inpainting methods can reconstruct smooth curves across large inpaint domains. This suggests that they should be effective for the cell path reconstruction problem and explore this idea in Chapter 4. We now conclude this chapter with a brief consideration of non-geometric classes of 2D inpainting techniques.

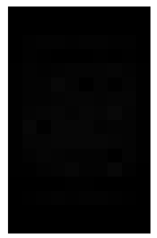

(a) u for LBIP

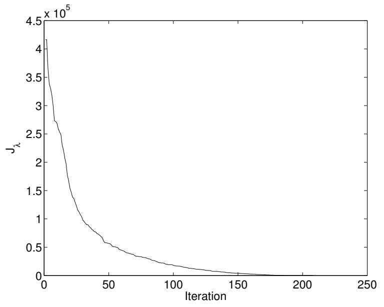


(b) J_λ for LBIP

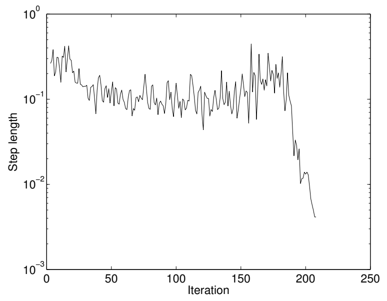


(c) Step lengths for LBIP

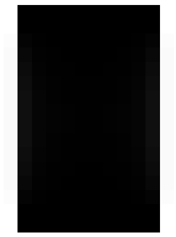

(d) u for SQP

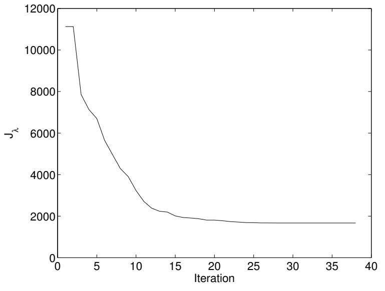


(e) J_λ for SQP

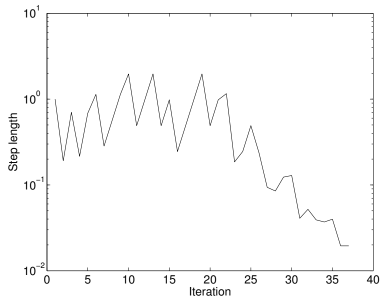


(f) Step lengths for SQP


(g) u for RTR

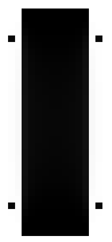


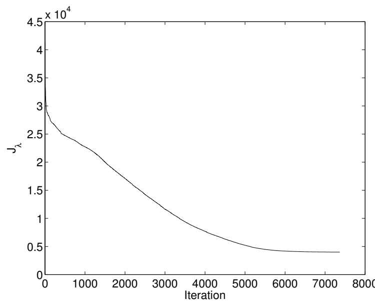
(h) J_λ for RTR



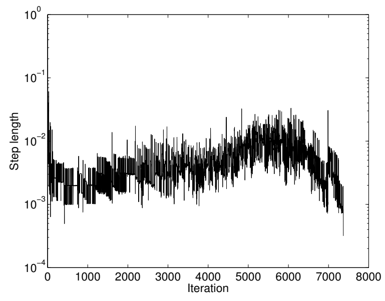
(i) Step lengths for RTR

Figure 3.16: Elastica by optimization: results on $w \times h = 10 \times 12$ black bar

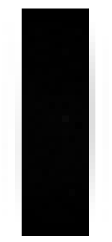

(a) u for LBIP

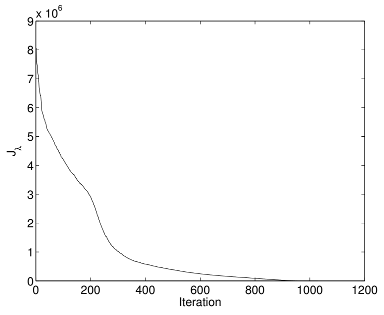


(b) J_λ for LBIP

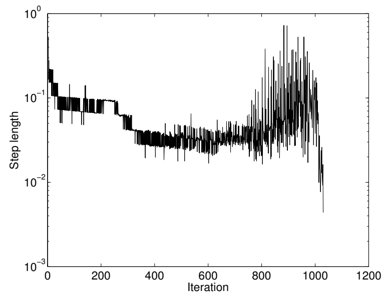


(c) Step lengths for LBIP

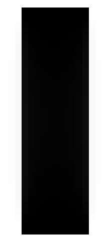

(d) u for SQP

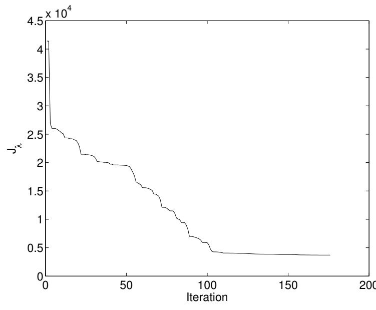


(e) J_λ for SQP

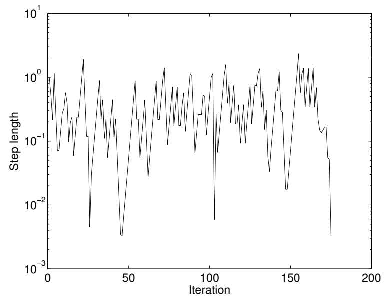


(f) Step lengths for SQP


(g) u for RTR



(h) J_λ for RTR



(i) Step lengths for RTR

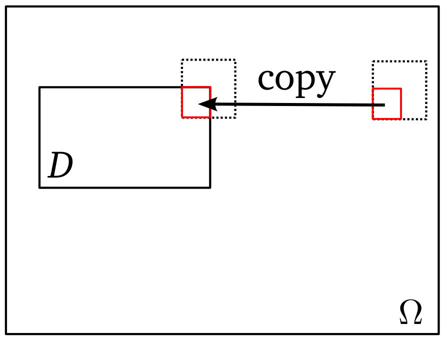
Figure 3.17: Elastica by optimization: results on $w \times h = 10 \times 30$ black bar

3.2 Inpainting by Texture Synthesis

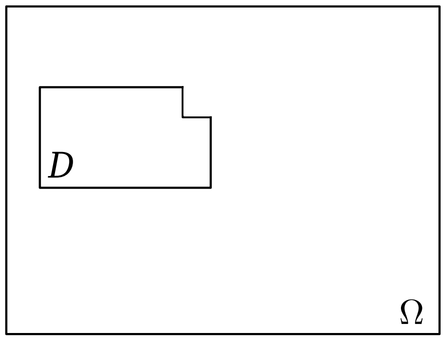
Any geometric model—including elastica—that is based on the idea of continuing level lines through D can only preserve geometry and cannot recover texture. The family of texture synthesis techniques has been developed to solve the problem of texture recovery.

Igehy and Pereira [30] recover a missing region in an image by filling it with a synthesized texture that matches the texture in the region surrounding the missing region. They modify the pyramid-based texture synthesis method of Heeger and Bergen [29] by adding a “composition step” that smoothly blends the synthesized texture with the existing image. The method is effective when the same texture surrounds all of D, but produces undesirable results when D covers visibly distinct textures. Also, the texture synthesis method uses a stochastic texture model that is incapable of recovering textures with regular patterns. A final difficulty with this method is that it only recovers texture and is unable to recover geometry inside the inpaint domain.

Bertalmio et al. [5] propose a method that recovers geometry and texture. They decompose the image u into its geometric, or structural, component v and its texture component w such that $u = v + w$. The v component is a cartoon-like approximation of u that contains smooth regions separated by sharp edges and is inpainted using Bertalmio et al.'s edge advection method (Section 3.1.1). The w component is reconstructed using an exemplar method (Section 3.3). The inpainted results of v and w are added to give the recovered solution. This method recovers both geometry and texture, but can produce artifacts in the result when the texture recovered for w does not correctly align with the structure recovered for v. The authors also report that the texture recovery step can produce incorrect textures when w contains textures at different scales. They propose a decomposition into multiple texture images, one for each scale. Another approach to recover texture is to fill the inpaint domain using segments copied from outside the inpaint domain. These segments could be taken from $\Omega \backslash D$ or from a library of different images. This is the idea behind exemplar and patch techniques.



(a) Step 1: copy a patch



(b) Step 2: update D

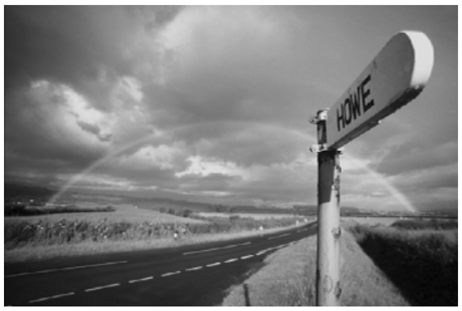
Figure 3.18: An exemplar method

3.3 Exemplar and Patch Techniques

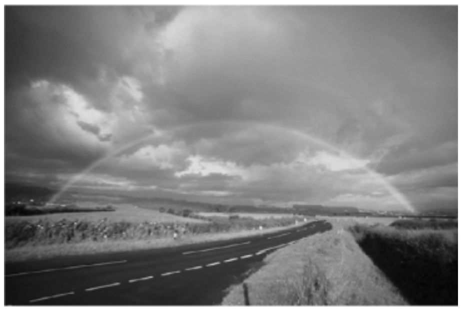
Geometric methods excel at recovering geometry, but they are local. The PDE is solved over D using only information at the boundary of D, while information on the rest of Ω is ignored. Bertalmio et al.'s texture inpainting method recovers texture globally, but the geometric portion of the image is still reconstructed using local information. Exemplar and patch methods address this concern. They fill the inpaint domain by copying patches from u or from a library of images into the inpaint domain D. The underlying assumption is that patches taken from realistic images already have a natural texture and structure so, if they are fitted together in a realistic way, the result will look natural [20].

In Criminis et al.'s exemplar method [20], information is propagated into D by selecting a patch from $\Omega \backslash D$ that is similar to a region near D. Similarity is measured by a sum of squared differences (SSD). Information from near the patch is then copied into the inpaint domain and the inpaint domain is updated to exclude the copied region (see Figure 3.18). This procedure is repeated until the inpaint domain has been filled. The method produces striking results, such as those shown in Figure 3.19, and can inpaint multiple textures and recover realistic geometry in many cases. The results, however, are dependent on the order in which patches are propagated.

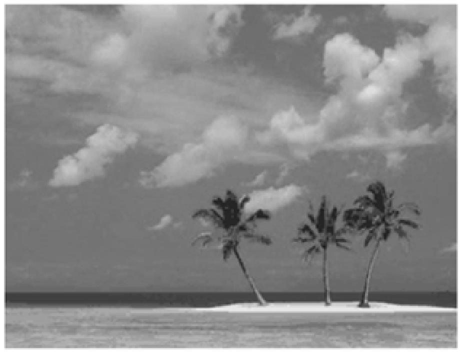
Cao et al. [14] attempt to overcome this problem by selecting patches that closely



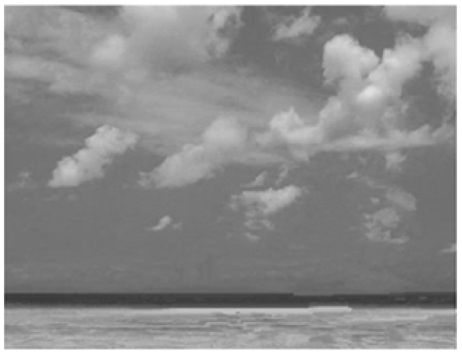
(a) Original image



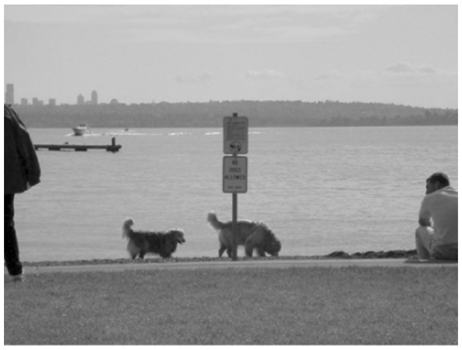
(b) Ipainting result



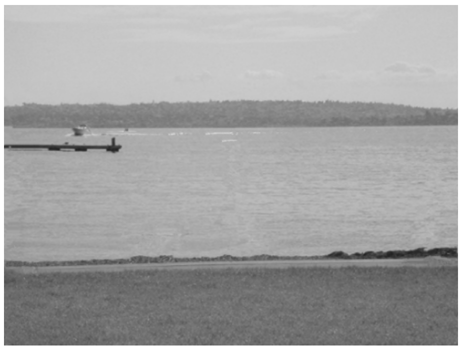
(c) Original image



(d) Ipainting result



(e) Original image



(f) Ipainting result

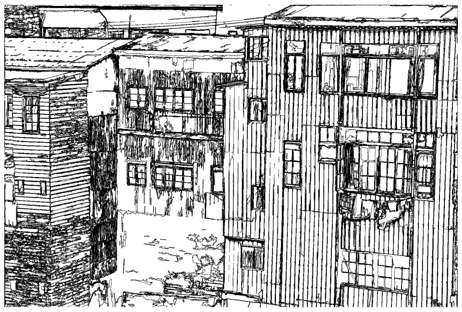
Figure 3.19: Results from Criminis et al.'s method (taken from [20])

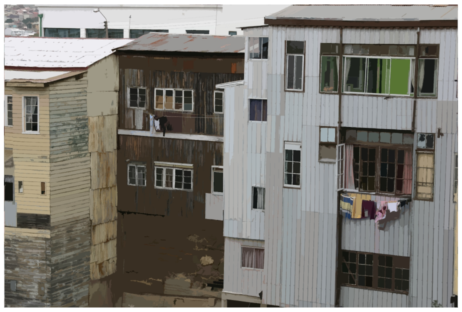
match a geometric sketch of the inpaint domain D. The sketch is a piecewise constant approximation of u. They compute the sketch by first extracting the meaningful level lines of u on $\Omega\backslash D$. Meaningful level lines are "level lines having a contrast that is very unlikely to be encountered in a white noise image" [14, p. 5]. They approximately correspond to edges, but exclude spurious edges introduced by noise and textures. Next the sketch is inpainted using a method similar to the dynamic programming approach of Masnou (Section 2.2.6). Level lines intersecting the boundary ∂D are heuristically matched to nearby level lines such that no level lines may intersect. These lines are reconnected using a curve model that is chosen to suit the problem. Cao et al. experiment with straight lines and Euler spirals. To complete the sketch, the regions defined by the meaningful level lines are filled with the average intensity value over that region in the original image. Finally, the sketch of D is used to guide an exemplar based-method. This process is depicted in Figure 3.20.

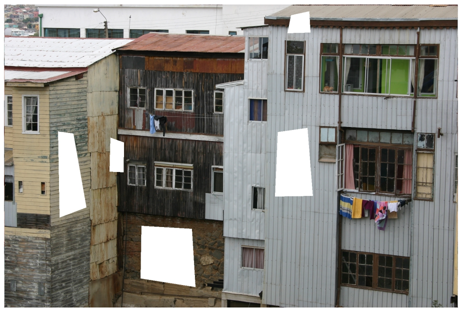
Cao et al.'s [14] method is similar to Crimini et al.'s [20], but when comparing candidate patches to a region at the boundary of the inpaint domain, the SSD is computed over the sketch of D as well as the region outside of D. This method produces excellent results on challenging inpainting problems in real photographs. It considers geometry as well as global image data when reconstructing D, but does not suffer from the problem of misalignment of texture and structure components seen in Bertalmio et al.'s texture inpainting method (Section 3.2). However, since the image is reconstructed using small patches, these matches may be copied in a configuration that minimizes SSD but looks unrealistic (imagine a human face reconstructed with eyes, nose and mouth positioned incorrectly).

Hays and Efros [28] guarantee a realistic result inside the inpaint domain by replacing all of D by a single patch copied from another image. Patches are selected from a database of over 2.3 million unique images. They select an initial list of candidate images using a database query that matches on a small set of semantic descriptors. From that candidate set, they select the image that is most similar to u (by SSD) in an 80-pixel band around D. Values for D are then copied from the chosen candidate image. This method has the desirable property of guaranteeing a natural result inside D as long as the image library

(a) Original image


(b) Meaningful level lines


(c) Cartoon approximation


(d) Test case

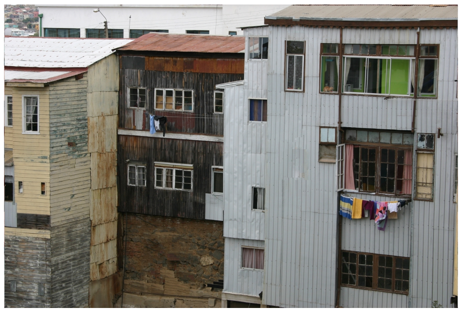

(e) Inpainting result

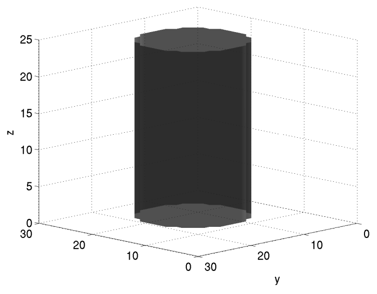
Figure 3.20: Illustration of Cao et al.'s method (taken from [14])

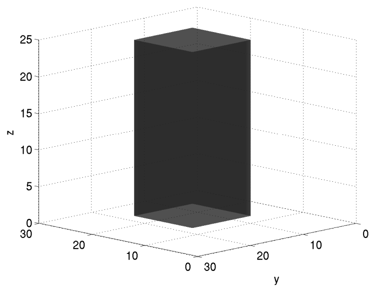
contains natural images. There is no guarantee, however, that the recovered D will be consistent with $\Omega \backslash D$. The method also requires a large and diverse image library. The library used to generate the results report in the paper consumes 396 GB of disk space. A final difficulty with this method is that to use it one must first acquire 2.3 million images.

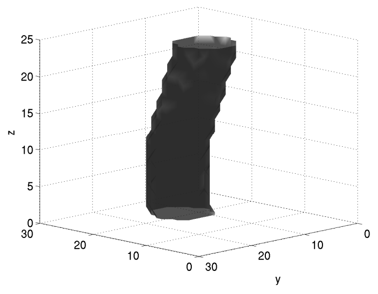
Chapter 4

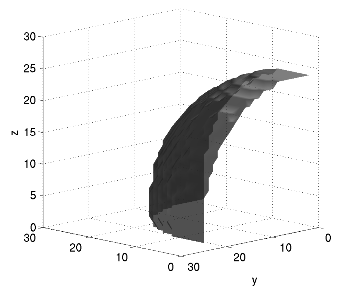
Geometric Models for 3D Inpainting

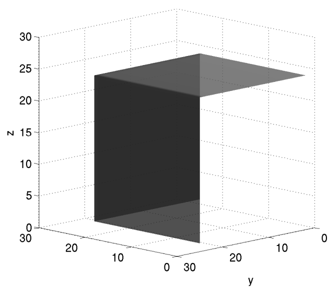
The TV, CDD, and elastica models discussed in Section 3.1 are formulated generally enough to allow direct application of the model to a 3D domain. This fact is mentioned in some inpainting papers, but it appears that no numerical results have been reported. Clarenz et al. [19] use inpainting models to recover damaged surfaces in $\mathbb{R}^3$, but the model applies only to smooth surfaces and cannot be applied to arbitrary volumes. We consider the direct application of the geometric models of Section 3.1 to 3D inpainting in a volume and illustrate properties of these models using the test cases depicted in Figure 4.1. In the original volumes, objects have intensity value 1 and empty space has intensity value 0. For viewing in 3D, we present the 0.99 level surfaces in all cases. Figure 4.2 shows the 3D cases after values in the inpaint domain have been replaced by the initial value 0.5. In all cases, D consists of the pixels that are a distance of at least four pixels from any side of the image volume. The cylinder and tube test cases are most relevant to cell path inpainting. The cylinder is an idealized cell path inpainting problem where the cell is perfectly circular and does not move over time. The tube simulates a moving cell.

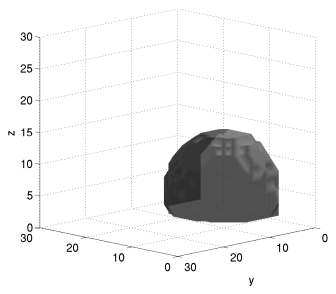

(a) Cylinder

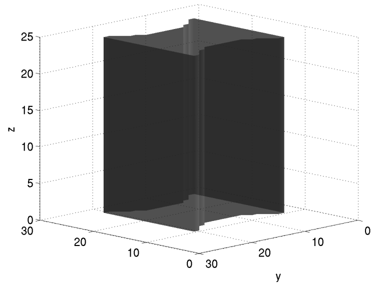

(b) Box

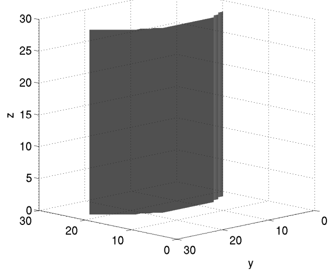

(c) Tube


(d) Corner sphere


(e) Corner cube


(f) Quarter sphere


(g) Hourglass


(h) Concave sheet

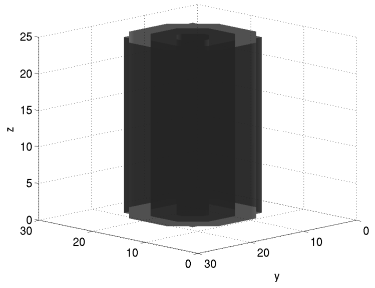
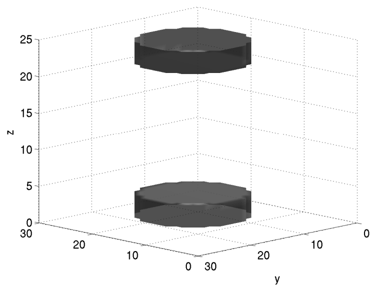
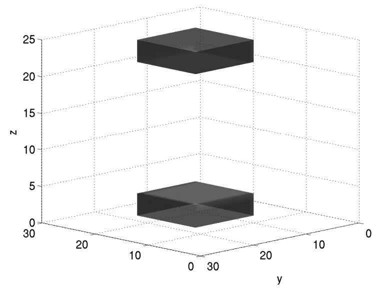

(i) Concentric cylinders

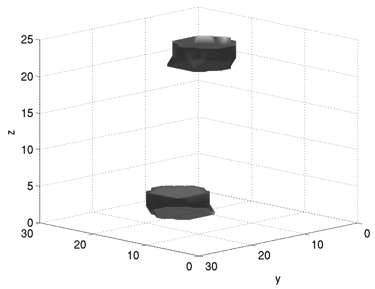
Figure 4.1: 3D test cases



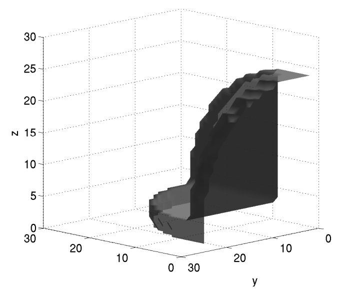
(a) Cylinder



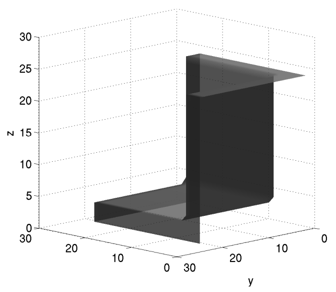
(b) Box



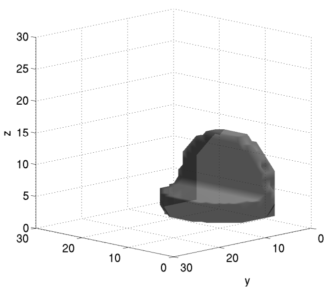
(c) Tube



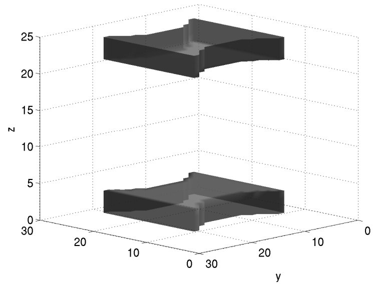
(d) Corner sphere



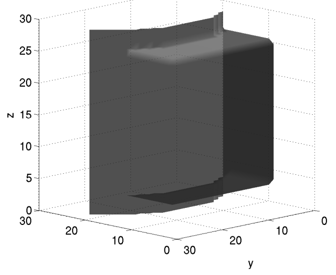
(e) Corner cube



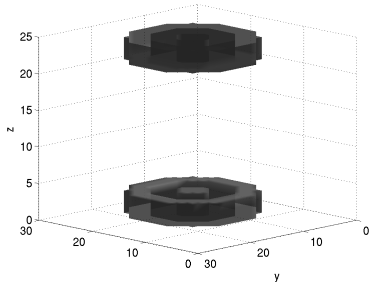
(f) Quarter sphere



(g) Hourglass



(h) Concave sheet



(i) Concentric cylinders

Figure 4.2: 3D test cases after setting values in D to 0.5

4.1 3D Total Variation

To apply TV inpainting in 3D, a third dimension is added to the domains Ω and D and to the corresponding integrals in Equation (3.4):

$$\underset{u}{\operatorname{argmin}} \int_{\Omega} |\nabla u| dxdydz + \frac{\lambda}{2} \int_{\Omega \setminus D} (u^0 - u)^2 dxdydz. \tag{4.1}$$

TV is zero on constant regions and non-zero on surfaces that form boundaries between constant regions, so the TV model will choose the solution of minimum surface area. This is similar to the 2D case where TV favours the solution of minimum perimeter. For the cylinder test case, the model will either leave the cylinder broken or reconnect it across D. If the cylinder is broken, then the total surface area inside D is that of the two disks that intersect ∂D: $2\pi r^2$ for a cylinder of radius r. If the surface is reconnected, then the total surface area inside D is the minimal surface of revolution that joins the two disks. It can be shown that the surface area is minimized by reconnecting the cylinder across D when

$$h/r < 1.325\ldots \tag{4.2}$$

where h is the height of the gap. Otherwise, the surface area of the two disks on ∂D is minimal and the cylinder will be left broken [49]. This is illustrated in Figure 4.3 where 3D TV inpainting is applied to cylinders with various gap height to radius ratios.

This inability to recover shapes across large gaps in D makes TV inpainting ill-suited for cell path reconstruction. Due to its ability to recover shapes across large inpaint domains and its strong preference for smooth curves, the elastica inpainting model seems better suited to the problem of cell path inpainting. We examine it next.

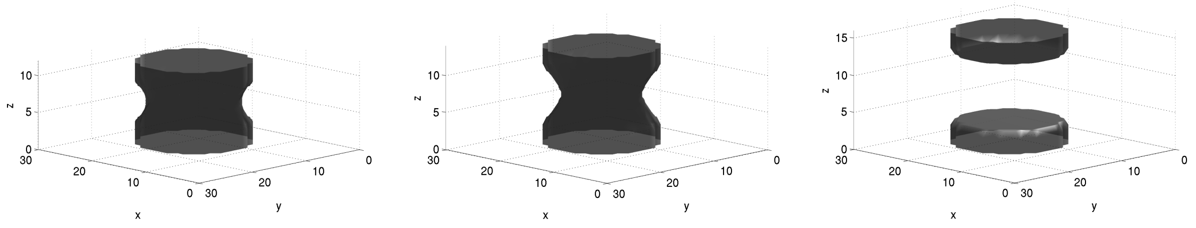

(a) $h/r = 0.5$ (b) $h/r = 0.75$ (c) $h/r = 1$

Figure 4.3: 0.99 level surfaces for 3D TV inpainting on the cylinder test case

4.2 3D Elastica Using Mean Curvature

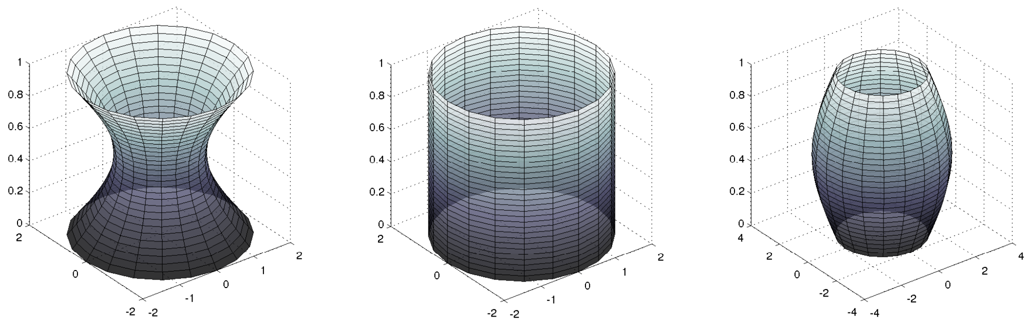
Directly generalizing the elastica model from Equation (3.10) gives

$$J_\lambda(u) = \int_\Omega (a + b\kappa^2)|\nabla u| dxdydz + \frac{\lambda}{2} \int_{\Omega \setminus D} (u^0 - u)^2 dxdydz. \tag{4.3}$$

In the 2D models from Section 3.1.4, curvature κ is typically computed using the divergence form $\kappa = \nabla \cdot \frac{\nabla u}{|\nabla u|}$. For a 2D image, this expression gives the curvature of the level line passing through a point. For a 3D volume, it gives the *mean curvature* of the level *surface* passing through a point. Applying this model to the cylinder reconstruction problem improves on the TV results—the model is able to reconstruct the object across wide gaps—but the shape of the reconstruction is not as desired. The 0.99 level surfaces for the cylinder and tube test cases are shown in Figures 4.7(a) and 4.7(g) on page 74.

4.2.1 Results

The result for the cylinder case is understood by considering the definition of the mean curvature κ. Consider a point p on the interior of a smooth surface. The intersection of the surface and a plane containing the unit vector normal to the surface at p forms a curve in $\mathbb{R}^3$. Taking the maximum and minimum curvatures of this curve at p over all such planes gives κ_1 and κ_2: the principle curvatures [26, 47]. Then mean curvature is defined as the



(a) $\kappa_{\mathrm{m}} \approx 0$, $\kappa_{\mathrm{g}} < 0$ (b) $\kappa_{\mathrm{m}} > 0$, $\kappa_{\mathrm{g}} = 0$ (c) $\kappa_{\mathrm{m}} > 0$, $\kappa_{\mathrm{g}} > 0$

Figure 4.4: Mean and Gaussian curvature on surfaces

average of the principle curvatures:

$$\kappa_{\mathrm{m}} = \frac{1}{2}\left(\kappa_1 + \kappa_2\right). \tag{4.4}$$

In the shapes in Figure 4.4, at points on the surfaces where grid lines intersect, the grid lines correspond to the lines of principle curvature. Consider computing mean curvature at one of these points of intersection in Figure 4.4(a). We have $\kappa_1 > 0$ corresponding to the horizontal grid lines, $\kappa_2 < 0$ corresponding to the vertical grid lines and $\kappa_{\mathrm{m}} = (\kappa_1+\kappa_2)/2 \approx 0$. For the cylinder in Figure 4.4(b), we have $\kappa_1 > 0$ corresponding to the horizontal grid lines, $\kappa_2 = 0$ corresponding to the vertical grid lines, and $\kappa_{\mathrm{m}} = (\kappa_1 + \kappa_2)/2 > 0$. Therefore, a model that minimizes squared mean curvature over the surface cannot recover a cylinder. As a result, although it reconnects the broken cylinder over a large D, the elastica model using mean curvature is not suitable for cell path inpainting. However, there is another definition of surface curvature better suited to this purpose. We apply the Gaussian curvature to cell path inpainting after considering the solution of 3D mean curvature elastica using numerical optimization.

	LBIP	**SQP**	**RTR**
a	0.5	0.5	0.5
b	20	20	20
Δx	1/16	1/16	1/16
ε	10^{-2}	10^{-1}	10^{-2}
σ	10^{-2}	10^{-2}	10^{-2}
curvature	minmod	fully expanded	minmod
tolx	10^{-4}	10^{-3}	10^{-3}
tolfun	$\mathbf{10^{-6}}$	10^{-2}	10^{-5}
tolcon	10^{-2}	10^{-2}	10^{-2}
maxiter	∞	∞	∞
maxfunevals	∞	∞	∞
hessian	BFGS	n/a	fin-diff-grads

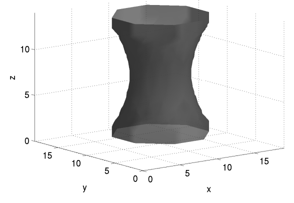
Table 4.1: Parameters for Matlab optimization of 3D elastica

	LBIP	**SQP**	**RTR**
CPU Time (minutes)	188.2	35791.2	2387.1
Iterations	6574	1954	271
# J_λ evaluations	10460	33109	272

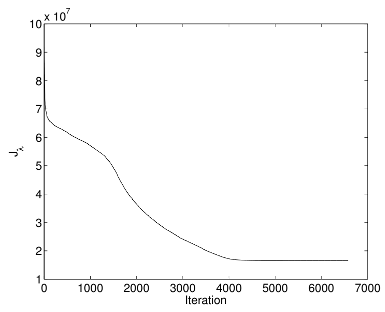
Table 4.2: Optimization time comparison for 3D elastica on $r \times h = 5.5 \times 10$ cylinder in a $19 \times 19 \times 14$ domain

4.2.2 Solution by Numerical Optimization

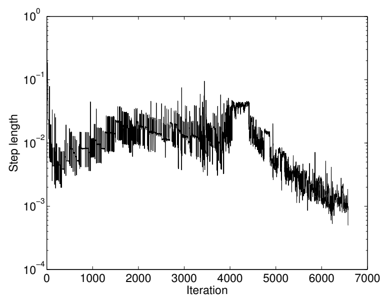
We now demonstrate that the numerical optimization techniques applied to 2D elastica in Section 3.1.4 are also effective for 3D elastica. We use similar parameters for the elastica model and optimization routines. They are listed in Table 4.1 where boldface values are different from those used in the 2D elastica optimization. Execution times are given in Table 4.2 and numerical results are shown in Figure 4.5. Notice how the non-zero assignment to the surface area parameter $a = 0.5$ pulls the surface inward more severely than $a = 0$ as used in Figure 4.7.



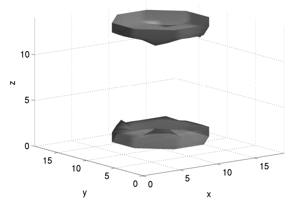
(a) u for LBIP



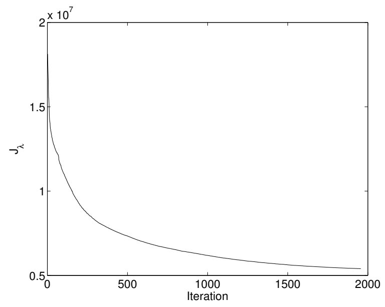
(b) J_λ for LBIP



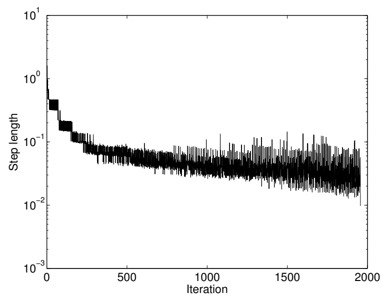
(c) Step lengths for LBIP



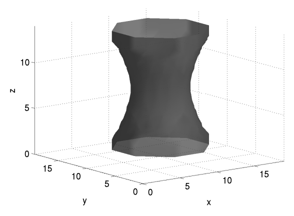
(d) u for SQP



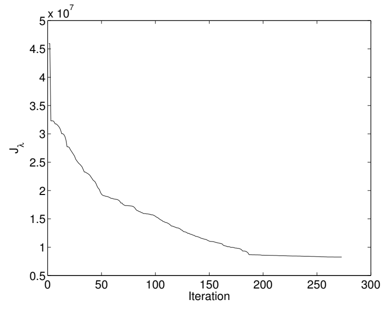
(e) J_λ for SQP



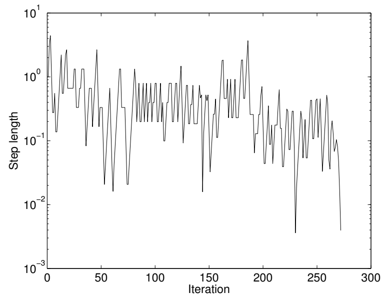
(f) Step lengths for SQP



(g) u for RTR



(h) J_λ for RTR



(i) Step lengths for RTR

Figure 4.5: 3D Elastica by optimization: results on $r \times h = 5.5 \times 10$ cylinder

4.3 3D Elastica Using Gaussian Curvature

We consider the Gaussian curvature of a surface to develop a new inpainting model based on elastica. Gaussian curvature is defined as the product of the principle curvatures:

$$\kappa_{\mathrm{g}} = \kappa_1 \kappa_2. \tag{4.5}$$

Since we have $\kappa_2 = 0$ for a point on the surface of the cylinder in Figure 4.4(b), we also have $\kappa_{\mathrm{g}} = 0$ for any such point. For the surface in Figure 4.4(a), we have $\kappa_1 > 0, \kappa_2 < 0$, and $\kappa_{\mathrm{g}} < 0$. For the surface in Figure 4.4(c), we have $\kappa_1 > 0, \kappa_2 > 0$, and $\kappa_{\mathrm{g}} > 0$. This suggests that we may recover cylinders using a model that minimizes squared Gaussian curvature over the recovered surface [26].

We balance the penalization of surface area and Gaussian curvature of the surface by setting $\kappa = \kappa_{\mathrm{g}}$ in the 3D elastica inpainting model. Although theoretically appealing, this model presents practical difficulties. The PDE given by the Euler-Lagrange equation is fourth-order, non-linear, and difficult to solve numerically:

$$0 = -\nabla \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} \tag{4.6}$$

where

$$V_1 = (a + b\kappa_{\mathrm{g}}^2)\frac{u_x}{|\nabla u|} - 2b\nabla\left(\kappa_{\mathrm{g}}|\nabla u|\right)\cdot\begin{bmatrix}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{xx}}\\ \frac{1}{2}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{xy}}\\ \frac{1}{2}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{xz}}\end{bmatrix}$$

$$V_2 = (a + b\kappa_{\mathrm{g}}^2)\frac{u_y}{|\nabla u|} - 2b\nabla\left(\kappa_{\mathrm{g}}|\nabla u|\right)\cdot\begin{bmatrix}\frac{1}{2}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{xy}}\\ \frac{\partial\kappa_{\mathrm{g}}}{\partial u_{yy}}\\ \frac{1}{2}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{yz}}\end{bmatrix}$$

$$V_3 = (a + b\kappa_{\mathrm{g}}^2)\frac{u_z}{|\nabla u|} - 2b\nabla\left(\kappa_{\mathrm{g}}|\nabla u|\right)\cdot\begin{bmatrix}\frac{1}{2}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{xz}}\\ \frac{1}{2}\frac{\partial\kappa_{\mathrm{g}}}{\partial u_{yz}}\\ \frac{\partial\kappa_{\mathrm{g}}}{\partial u_{zz}}\end{bmatrix}.$$

We compute Gaussian curvature in 3D using the formula [48]:

$$\kappa_{\mathrm{g}} = \frac{t_1 t_2 - t_3^2}{t_4^2} \tag{4.7}$$

where

$$\begin{aligned}
t_1 &= u_z\left(u_{xx}u_z - 2u_xu_{xz}\right) + u_x^2u_{zz}\\
t_2 &= u_z\left(u_{yy}u_z - 2u_yu_{yz}\right) + u_y^2u_{zz}\\
t_3 &= u_z\left(-u_xu_{yz} + u_{xy}u_z - u_{xz}u_y\right) + u_xu_yu_{zz}\\
t_4 &= u_z\left(u_x^2 + u_y^2 + u_z^2\right).
\end{aligned}$$

We attempt to solve this PDE using artificial time marching and the forward Euler method (Section 2.2), but the time step is too restricted for the scheme to be of any practical use. Applying the semi-implicit splitting of Brito-Loeza and Chen [10] produces an unstable numerical scheme.

4.3.1 Instability in the Numerical Schemes

To understand the instability, we consider the elastica term in the energy functional (4.3). By the coarea formula [2, 25], this integral is equivalent to the area integral over level surfaces in the volume:

$$\int_{\Omega} \left(a + b\kappa_{\mathrm{g}}^2\right) |\nabla u| dxdydz = \int_0^1 \int_{u=r} a + b\kappa_{\mathrm{g}}^2 \; dAdr.$$

Since a and b are constant and $\kappa_{\mathrm{g}} = \kappa_1\kappa_2$, this integral is minimized when either of κ_1 or κ_2 is zero. If, for example, $\kappa_1 = 0$ then $|\kappa_2|$ may be arbitrarily large. We therefore introduce an additional regularization term to ensure that both $|\kappa_1|$ and $|\kappa_2|$ are bounded.

To avoid computing the principle curvatures directly, we reintroduce the minimization of squared mean curvature:

$$\int_0^1 \int_{u=r} a + b\kappa_{\mathrm{m}}^2 + c\kappa_{\mathrm{g}}^2 \; dAdr.$$

We expand curvatures in terms of the principle curvatures and complete the square to verify that both principle curvatures are minimized:

$$a + b\left(\frac{\kappa_1 + \kappa_2}{2}\right)^2 + c\left(\kappa_1\kappa_2\right)^2 = c\left(\kappa_1\kappa_2 + \frac{b}{4c}\right)^2 - \frac{b^2}{16c} + a + \frac{b}{4}\kappa_1^2 + \frac{b}{4}\kappa_2^2.$$

4.4 3D Elastica Using Mean and Gaussian Curvature

We find that reintroducing mean curvature to the Gaussian curvature elastica model yields a numerically solvable minimization. Therefore, we propose the mean/Gaussian elastica energy functional:

$$J_\lambda(u) = \int_{\Omega} (a + b\kappa_{\mathrm{m}}^2 + c\kappa_{\mathrm{g}}^2)|\nabla u| dxdydz + \frac{\lambda}{2}\int_{\Omega\setminus D} (u^0 - u)^2 dxdydz \qquad (4.8)$$

where the desired solution is

$$\underset{u}{\operatorname{argmin}} J_\lambda(u). \tag{4.9}$$

Since this model is the main result of this work, we cover in detail the numerical scheme that we use to solve it.

4.4.1 Solution by Numerical Optimization

The effectiveness of the numerical optimization techniques for mean curvature elastica leads us to apply the same methods to mean/Gaussian elastica. We use the parameters given in Table 4.3 where boldface values differ from those used for 3D mean curvature elastica. Unfortunately, the optimization techniques terminate after few iterations without any visible change in u. In all cases, the scheme terminates because the relative change in all elements of u is less than `tolx`.

The results in Table 4.4 and Figure 4.6 demonstrate that the optimization techniques as we have applied them are not effective for solving the mean/Gaussian elastica model in 3D. These results might be improved by using an analytically computed Hessian matrix for the interior point and reflective trust region methods, but manually computing the Hessian of the mean/Gaussian elastica minimization manually is complicated and error-prone. We attempt to compute the Hessian using the symbolic approach that we use for the gradient, but the program exhausts our computer's memory. This leads us once again to the fixed point splitting method.

4.4.2 Manipulations of the Euler-Lagrange Equation

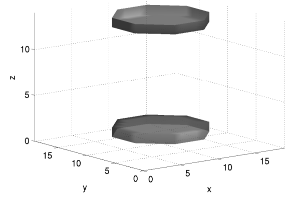
We apply the calculus of variations to the minimization (4.9) and solve the Euler-Lagrange equation directly by applying the fixed point method proposed by Brito-Loeza and Chen [10] for 2D elastica inpainting. Their stability result, quoted in Theorem 1, is based on the gradient flow splitting methods introduced in Section 2.2.3 and 2.2.4:

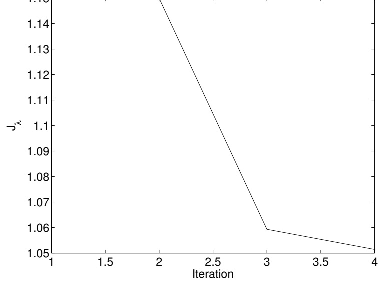
	LBIP	**SQP**	**RTR**
a	0.5	0.5	0.5
b	**10**	**10**	**10**
c	**125**	**125**	**125**
Δx	1/16	1/16	1/16
ε	10^{-2}	10^{-1}	10^{-2}
σ	10^{-2}	10^{-2}	10^{-2}
curvature	minmod	fully expanded	minmod
tolx	$\mathbf{10^{-14}}$	$\mathbf{10^{-14}}$	$\mathbf{10^{-12}}$
tolfun	10^{-6}	10^{-2}	10^{-5}
tolcon	10^{-2}	10^{-2}	10^{-2}
maxiter	∞	∞	∞
maxfunevals	∞	∞	∞
hessian	BFGS	n/a	fin-diff-grads

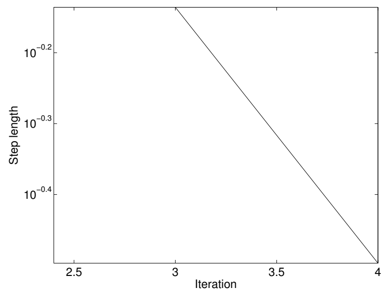
Table 4.3: Parameters for Matlab optimization of 3D mean/Gaussian elastica

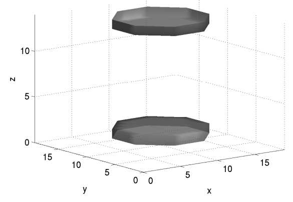
	LBIP	**SQP**	**RTR**
CPU Time (minutes)	0.3	130.0	232.3
Iterations	3	5	25
# J_λ **evaluations**	55	64	26

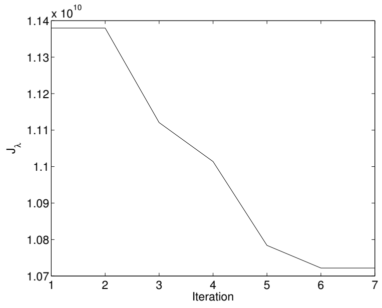
Table 4.4: Optimization time comparison for 3D mean/Gaussian elastica on $r \times h = 5.5 \times 10$ cylinder

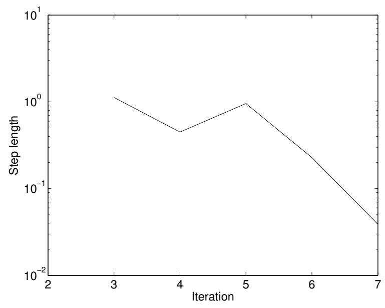

(a) u for LBIP

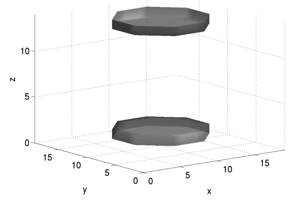

(b) J_λ for LBIP

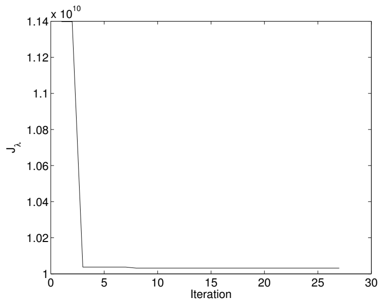

(c) Step lengths for LBIP


(d) u for SQP


(e) J_λ for SQP


(f) Step lengths for SQP


(g) u for RTR


(h) J_λ for RTR

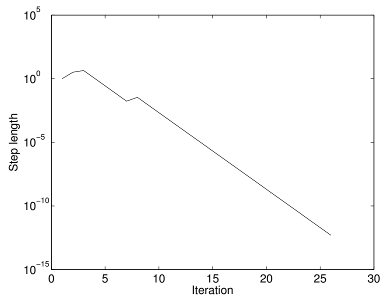

(i) Step lengths for RTR

Figure 4.6: 3D mean/Gauss Elastica optimization: results on $r \times h = 5.5 \times 10$ cylinder

Theorem 1. *Suppose*

$$\begin{cases} \frac{\partial u}{\partial t} = -\nabla_f J(u) \\ u(0) = u^0 \end{cases}$$

is a gradient system by Definition 1 where $J(u) = \int_\Omega f(u) d\vec{x}$ *is an energy functional defined over functions* $u : \Omega \to \mathbb{R}$ *and* J *may be split such that*

$$J = J_{11} - J_{12}^A - J_{12}^B - J_2$$
$$J_{12}^A = \int_\Omega g_1(u) g_2(u) d\vec{x}$$

with J_{11}*,* J_{12}*,* J_{12}^B*, and* J_2 *being strictly convex. Then, for any* u^0 *and sufficiently large* $C_1 > 0$ *as defined below, the numerical scheme*

$$\begin{aligned} \frac{u^{n+1} - u^n}{\Delta t} &= -\nabla_f J_{11}(u^{n+1}) \\ &\quad + g_2(u^n) \nabla_f \left(\int_\Omega g_1(u^n) d\vec{x} \right) + g_1(u^{n+1}) \nabla_f \left(\int_\Omega g_2(u^{n+1}) d\vec{x} \right) \\ &\quad + \nabla_f J_{12}^B(u^k) - \nabla_f J_2(u^{n+1}) \end{aligned} \tag{4.10}$$

is gradient stable for all positive Δt*.*

Since for digital inpainting we are only concerned with the final result and not with maintaining accuracy over time, we may construct a fixed point method to find the steady state solution of Equation 4.10 as follows:

$$\frac{u^{n+1} - u^n}{\Delta t} = 0 = -\nabla_f J. \tag{4.11}$$

Brito-Loeza and Chen [10] conjecture that this scheme inherits unconditional stability from Theorem 1, but note that they have not yet found a proof of this claim.

We now split Equation 4.8 according to Theorem 1. We split J_λ so that $J_\lambda = J_1 + J_2$:

$$J_1 = \int_\Omega (a + b\kappa_{\mathrm{m}}^2 + c\kappa_{\mathrm{g}}^2) |\nabla u| dx dy dz$$

and

$$J_2 = \frac{\lambda}{2} \int_{\Omega \setminus D} (u^0 - u)^2 dxdydz.$$

Since the function $f(x) = x^2$ is convex and the integral preserves convexity, J_2 is convex [8]. Next, we split J_1 into J_{11} and J_{12} so that $J_1 = J_{11} - J_{12}$:

$$J_{11} = (a + C_1) \int_{\Omega} |\nabla u| dxdydz$$

and

$$J_{12} = \int_{\Omega} \left(-b\kappa_{\mathrm{m}}^2 - c\kappa_{\mathrm{g}}^2 \right) |\nabla u| dxdydz + C_1 \int_{\Omega} |\nabla u| dxdydz$$

where C_1 is a constant parameter. J_{11} is convex since

$$\begin{aligned} J_{11}(\alpha u + \beta v) &= (a + C_1) \int_{\Omega} |\nabla \left(\alpha u + \beta v \right)| dxdydz \\ &\leq (a + C_1) \int_{\Omega} \alpha |\nabla u| + \beta |\nabla v| dxdydz \\ &= \alpha (a + C_1) \int_{\Omega} |\nabla u| dxdydz + \beta (a + C_1) \int_{\Omega} |\nabla v| dxdydz \\ &= \alpha J_{11}(u) + \beta J_{11}(v) \end{aligned}$$

where $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$ [8]. J_{12} is convex provided the parameter C_1 is chosen sufficiently large. Finally, we split J_{12} so that $J_{12} = J_{12}^A + J_{12}^B$ with $J_{12}^A = \int_{\Omega} g_1 g_2 dxdydz$:

$$g_1 = -b\kappa_{\mathrm{m}}^2 - c\kappa_{\mathrm{g}}^2$$

$$g_2 = |\nabla u|$$

$$J_{12}^B = C_1 \int_{\Omega} |\nabla u| dxdydz.$$

Computing the gradient of each term gives:

$$\begin{aligned}
\nabla_f J_{11} &= -(a + C_1)\nabla \cdot \frac{\nabla u}{|\nabla u|} \\
\nabla_f J_{12}^A &= g_1 \nabla_f \left(\int_\Omega g_2 dxdydz \right) + g_2 \nabla_f \left(\int_\Omega g_1 dxdydz \right) \\
g_1 \nabla_f \left(\int_\Omega g_2 dxdydz \right) &= \nabla \cdot \left[\left((b\kappa_{\mathrm{m}}^2 + c\kappa_{\mathrm{g}}^2) \frac{\nabla u}{|\nabla u|} \right) \right] \\
g_2 \nabla_f \left(\int_\Omega g_1 dxdydz \right) &= \nabla \cdot \begin{bmatrix} V_1(u) \\ V_2(u) \\ V_3(u) \end{bmatrix} \\
\nabla_f J_{12}^B &= -C_1 \nabla \cdot \frac{\nabla u}{|\nabla u|} \\
\nabla_f J_2 &= -(u^0 - u)\lambda_{\Omega\backslash D}
\end{aligned}$$

where, as before,

$$\lambda_{\Omega\backslash D}(x) = \begin{cases} \lambda, & \text{if } x \in \Omega\backslash D \\ 0, & \text{otherwise} \end{cases},$$

the V terms are defined as

$$\begin{aligned}
V_1(u) &= \frac{2b}{|\nabla u|^3} \nabla\left(\kappa_{\mathrm{m}}|\nabla u|\right) \cdot \begin{bmatrix} -u_y^2 - u_z^2 \\ u_x u_y \\ u_x u_z \end{bmatrix} - 2b\nabla\left(\kappa_{\mathrm{g}}|\nabla u|\right) \cdot \begin{bmatrix} \frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xx}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xy}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xz}} \end{bmatrix} \\
V_2(u) &= \frac{2b}{|\nabla u|^3} \nabla\left(\kappa_{\mathrm{m}}|\nabla u|\right) \cdot \begin{bmatrix} u_x u_y \\ -u_x^2 - u_z^2 \\ u_y u_z \end{bmatrix} - 2b\nabla\left(\kappa_{\mathrm{g}}|\nabla u|\right) \cdot \begin{bmatrix} \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xy}} \\ \frac{\partial \kappa_{\mathrm{g}}}{\partial u_{yy}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{yz}} \end{bmatrix} \\
V_3(u) &= \frac{2b}{|\nabla u|^3} \nabla\left(\kappa_{\mathrm{m}}|\nabla u|\right) \cdot \begin{bmatrix} u_x u_z \\ u_y u_z \\ -u_x^2 - u_y^2 \end{bmatrix} - 2b\nabla\left(\kappa_{\mathrm{g}}|\nabla u|\right) \cdot \begin{bmatrix} \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xz}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{yz}} \\ \frac{\partial \kappa_{\mathrm{g}}}{\partial u_{zz}} \end{bmatrix},
\end{aligned}$$

and the derivatives of κ_{g} with respect to the partial derivatives of u are:

$$\begin{aligned}
\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xx}} &= \frac{u_{yy}u_z^2 - 2u_z u_y u_{yz} + u_y^2 u_{zz}}{|\nabla u|^4} \\
\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{yy}} &= \frac{u_{xx}u_z^2 - 2u_z u_x u_{xz} + u_x^2 u_{zz}}{|\nabla u|^4} \\
\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{zz}} &= \frac{u_{xx}u_y^2 + u_x^2 u_{yy} - 2u_{xy}u_x u_y}{|\nabla u|^4} \\
\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xy}} &= \frac{2u_z u_x u_{yz} - 2u_{xy}u_z^2 + 2u_z u_{xz} u_y - 2u_x u_y u_{zz}}{|\nabla u|^4} \\
\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xz}} &= \frac{-2u_z u_x u_{yy} + 2u_x u_y u_{yz} + 2u_{xy} u_z u_y - 2u_{xz} u_y^2}{|\nabla u|^4} \\
\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{yz}} &= \frac{-2u_{xx} u_z u_y + 2u_x u_{xz} u_y - 2u_x^2 u_{yz} + 2u_z u_x u_{xy}}{|\nabla u|^4}.
\end{aligned} \tag{4.12}$$

The calculations of $g_1 \nabla_f \int_\Omega g_2$ and $g_2 \nabla_f \int_\Omega g_1$ follow the proof of Theorem 5.1 in [17].

By substituting these equations into Equation (4.11), but not yet discretizing in space or time, we obtain the PDE that we solve numerically:

$$\begin{aligned}
0 = (a + C_1)\nabla \cdot \frac{\nabla u}{|\nabla u|} + \nabla \cdot \left[\left((b\kappa_{\mathrm{m}}^2 + c\kappa_{\mathrm{g}}^2) \frac{\nabla u}{|\nabla u|} \right) + \begin{bmatrix} V_1(u) \\ V_2(u) \\ V_3(u) \end{bmatrix} \right] \\
- C_1 \nabla \cdot \frac{\nabla u}{|\nabla u|} + (u^0 - u)\lambda_{\Omega \setminus D}.
\end{aligned} \tag{4.13}$$

subject to the boundary condition

$$u = u^0 \text{ on } \partial\Omega$$

4.4.3 Numerical Solution of the Manipulated PDE

To solve the model, we discretize Equation (4.13) according to Theorem 1 and linearize the left-hand side to apply a fixed point iteration. We assume that u is discritized on a uniform grid with spacing Δx and let subscripts of u denote spatial grid coordinates. We use D_x to represent a finite difference discretiztion of a partial derivative with respect to x. Since the PDE is fourth-order, we use midpoint finite differences to ensure a small stencil. We solve the PDE semi-implicitly using a fixed point iteration and let superscripts of u denote the iteration step. To obtain a fixed point method, we lag $1/|\nabla u|$ in the $g_1 \nabla_f \int_\Omega g_2$ term on the left-hand side to obtain constant coefficients for u^{n+1} at iteration n:

$$-\nabla \cdot \left(\left(a + C_1 + b\kappa_{\mathrm{m}}^2(u^n) + c\kappa_{\mathrm{g}}^2(u^n) \right) \frac{\nabla u^{n+1}}{|\nabla u^n|} \right) + u^{n+1} \lambda_{\Omega \setminus D} = r(u^n) \tag{4.14}$$

where

$$r(u^n) = \nabla \cdot \begin{bmatrix} V_1(u^n) \\ V_2(u^n) \\ V_3(u^n) \end{bmatrix} - C_1 \nabla \cdot \frac{\nabla u^n}{|\nabla u^n|} + u^0 \lambda_{\Omega \setminus D}.$$

Left-Hand Side

To simplify notation for the left-hand side of Equation (4.14), we define

$$m_{ijk}^n = \frac{a + C_1 + b\kappa_{\mathrm{m}}^2(u_{ijk}^n) + c\kappa_{\mathrm{g}}^2(u_{ijk}^n)}{\Delta x^2 |\nabla u_{ijk}^n|}.$$

The discretizations for curvatures and $|\nabla u|$ are defined below. Now, by expanding the gradient operator in Equation (4.14), we obtain:

$$-\nabla \cdot \left(\frac{a + C_1 + b\kappa_{\mathrm{m}}^2(u^n) + c\kappa_{\mathrm{g}}^2(u^n)}{\Delta x |\nabla u^n|} \begin{bmatrix} u_{(i+1/2)jk}^{n+1} - u_{(i-1/2)jk}^{n+1} \\ u_{i(j+1/2)k}^{n+1} - u_{i(j-1/2)k}^{n+1} \\ u_{ij(k+1/2)}^{n+1} - u_{ij(k-1/2)}^{n+1} \end{bmatrix} \right) + u_{ijk}^{n+1} \lambda_{\Omega \setminus D}.$$

Expanding the divergence operator gives:

$$\begin{aligned}
& m^n_{(i+1/2)jk}\left(u^{n+1}_{(i+1)jk} - u^{n+1}_{ijk}\right) - m^n_{(i-1/2)jk}\left(u^{n+1}_{ijk} - u^{n+1}_{(i-1)jk}\right) \\
& + m^n_{i(j+1/2)k}\left(u^{n+1}_{i(j+1)k} - u^{n+1}_{ijk}\right) - m^n_{i(j-1/2)k}\left(u^{n+1}_{ijk} - u^{n+1}_{i(j-1)k}\right) \\
& + m^n_{ij(k+1/2)}\left(u^{n+1}_{ij(k+1)} - u^{n+1}_{ijk}\right) - m^n_{ij(k-1/2)}\left(u^{n+1}_{ijk} - u^{n+1}_{ij(k-1)}\right) \\
& + u^{n+1}_{ijk}\lambda_{\Omega\setminus D}.
\end{aligned}$$

Then, by rearranging, we identify the coefficients of the grid points of u^{n+1}:

$$\begin{aligned}
m^n_{(i+1/2)jk}u^{n+1}_{(i+1)jk} &+ m^n_{i(j+1/2)k}u^{n+1}_{i(j+1)k} + m^n_{ij(k+1/2)}u^{n+1}_{ij(k+1)} \\
&+ m^n_{(i-1/2)jk}u^{n+1}_{(i-1)jk} + m^n_{i(j-1/2)k}u^{n+1}_{i(j-1)k} + m^n_{ij(k-1/2)}u^{n+1}_{ij(k-1)} \\
&- \big(m^n_{(i+1/2)jk} + m^n_{i(j+1/2)k} + m^n_{ij(k+1/2)} \\
&\quad + m^n_{(i-1/2)jk} + m^n_{i(j-1/2)k} + m^n_{ij(k-1/2)} - \lambda_{\Omega\setminus D}\big)\, u^{n+1}_{ijk}
\end{aligned}$$

To complete the discretization of the left-hand side, we must evaluate the m coefficients at grid midpoints. The m coefficients depend on mean curvature, Gaussian curvature, and first partial derivatives of u (in $|\nabla u|$). These are discretized below.

Right-Hand Side

We now consider the spatial discretization of $r(u^n)$ in the right-hand side of Equation (4.14). First, for the $C_1\nabla \cdot \frac{\nabla^n_{ijk}}{|\nabla u^n_{ijk}|}$ term, we note that this is the divergence form of mean curvature and that its discretization is discussed below. Spatially discretizing the divergence operator on the V term gives

$$\frac{1}{\Delta x}\begin{bmatrix} V_1(u^n_{(i+1/2)jk}) - V_1(u^n_{(i-1/2)jk}) \\ V_2(u^n_{i(j+1/2)k}) - V_2(u^n_{i(j-1/2)k}) \\ V_3(u^n_{ij(k+1/2)}) - V_3(u^n_{ij(k-1/2)}) \end{bmatrix}.$$

Discretizing the V terms at midpoints is similar for all dimensions, so we only include details for $V_1(u^n_{(i+1/2)jk})$:

$$
\begin{aligned}
&V_1(u^n_{(i+1/2)jk}) = \\
&\quad \frac{2b}{|\nabla u^n_{(i+1/2)jk}|^3} \nabla\left(\kappa_{\mathrm{m}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \cdot \begin{bmatrix} -(D_y u^n_{(i+1/2)jk})^2 - (D_z u^n_{(i+1/2)jk})^2 \\ (D_x u^n_{(i+1/2)jk})(D_y u^n_{(i+1/2)jk}) \\ (D_x u^n_{(i+1/2)jk})(D_z u^n_{(i+1/2+jk})\end{bmatrix} \\
&\qquad - 2b\nabla\left(\kappa_{\mathrm{g}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \cdot \begin{bmatrix} \frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xx}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xy}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xz}} \end{bmatrix}^n_{(i+1/2)jk}.
\end{aligned}
$$

Expanding gradients gives:

$$
\begin{aligned}
&V_1(u^n_{(i+1/2)jk}) = \\
&\quad \frac{2b}{|\nabla u^n_{(i+1/2)jk}|^3} \begin{bmatrix} D_x\left(\kappa_{\mathrm{m}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \\ D_y\left(\kappa_{\mathrm{m}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \\ D_z\left(\kappa_{\mathrm{m}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \end{bmatrix} \cdot \begin{bmatrix} -(D_y u^n_{(i+1/2)jk})^2 - (D_z u^n_{(i+1/2)jk})^2 \\ (D_x u^n_{(i+1/2)jk})(D_y u^n_{(i+1/2)jk}) \\ (D_x u^n_{(i+1/2)jk})(D_z u^n_{(i+1/2+jk})\end{bmatrix} \\
&\qquad - 2b \begin{bmatrix} D_x\left(\kappa_{\mathrm{g}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \\ D_y\left(\kappa_{\mathrm{g}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \\ D_z\left(\kappa_{\mathrm{g}}(u^n_{(i+1/2)jk})|\nabla u^n_{(i+1/2)jk}|\right) \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xx}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xy}} \\ \frac{1}{2}\frac{\partial \kappa_{\mathrm{g}}}{\partial u_{xz}} \end{bmatrix}^n_{(i+1/2)jk}.
\end{aligned}
$$

The midpoint discretizations for curvatures and partial derivatives of κ_{g} with respect to partial derivatives of u are defined below.

Curvatures

We use the divergence form of mean curvature discretized as:

$$\begin{aligned}\nabla \cdot \frac{\nabla u_{ijk}}{|\nabla u_{ijk}|} &= \frac{u_{(i+1)jk} - u_{ijk}}{\nabla u_{(i+1/2)jk}} - \frac{u_{ijk} - u_{(i-1)jk}}{\nabla u_{(i-1/2)jk}} \\ &+ \frac{u_{i(j+1)k} - u_{ijk}}{\nabla u_{i(j+1/2)k}} - \frac{u_{ijk} - u_{i(j-1)k}}{\nabla u_{i(j-1/2)k}} \\ &+ \frac{u_{ij(k+1)} - u_{ijk}}{\nabla u_{ij(k+1/2)}} - \frac{u_{ijk} - u_{ij(k-1)}}{\nabla u_{ij(k-1/2)}}.\end{aligned}$$

The formula for Gaussian curvature is given in Equation (4.7). It depends only on partial derivatives of u_{ijk}; these are explained below.

To compute curvatures at grid midpoints, we compute the minmod of the curvatures at the two nearest grid points:

$$\operatorname{minmod}(x, y) = \begin{cases} \operatorname{sgn}(x)\min(|x|, |y|) & \text{if } xy > 0 \\ 0 & \text{otherwise} \end{cases}.$$

For example,

$$\kappa_{\mathrm{m}}(u_{(i+1/2)jk}) = \operatorname{minmod}(\kappa_{\mathrm{m}}(u_{(i+1)jk}), \kappa_{\mathrm{m}}(u_{ijk})).$$

Partial Derivatives of Gaussian Curvature

The derivatives of Gaussian curvature, defined in Equations (4.12), depend only on first and second partial derivatives of u_{ijk}. We discuss these next.

Partial Derivatives of u

We compute first partial derivatives using centered differences and minmod:

$$\begin{aligned}
D_x u_{ijk} &= \frac{1}{2\Delta x}\left(u_{(i+1)jk} - u_{(i-1)jk}\right)\\
D_x u_{(i+1/2)jk} &= \frac{1}{\Delta x}\left(u_{(i+1)jk} - u_{ijk}\right)\\
D_x u_{i(j+1/2)k} &= \operatorname{minmod}(D_x u_{i(j+1)k}, D_x u_{ijk}).
\end{aligned}$$

We compute second partials at grid points using a standard three point stencil when both derivatives are taken along the same dimension and a four-point stencil for cross derivatives:

$$\begin{aligned}
D_{xx} u_{ijk} &= \frac{1}{\Delta x^2}\left(u_{(i+1)jk} - 2u_{ijk} + u_{(i-1)jk}\right)\\
D_{xy} u_{ijk} &= \frac{1}{\Delta x^2}\left(u_{(i+1)(j+1)k} - u_{(i-1)(j+1)k} - u_{(i+1)(j-1)k} + u_{(i-1)(j-1)k}\right).
\end{aligned}$$

Finally, to compute second derivatives and cross derivatives at midpoints, we use minmod:

$$\begin{aligned}
D_{xx} u_{(i+1/2)jk} &= \operatorname{minmod}(D_{xx} u_{(i+1)jk}, D_{xx} u_{ijk})\\
D_{xx} u_{i(j+1/2)k} &= \operatorname{minmod}(D_{xx} u_{i(j+1)k}, D_{xx} u_{ijk})\\
D_{xy} u_{(i+1/2)jk} &= \operatorname{minmod}(D_{xy} u_{(i+1)jk}, D_{xy} u_{ijk})\\
D_{xy} u_{ij(k+1/2)} &= \operatorname{minmod}(D_{xy} u_{ij(k+1)}, D_{xy} u_{ijk}).
\end{aligned}$$

Zeros in the Denominator

When $|\nabla u|$ appears in a denominator, it is computed as

$$|\nabla u| \approx \sqrt{u_x^2 + u_y^2 + u_z^2 + \varepsilon}$$

for a small constant ε. This prevents division by zero in constant regions of the volume. Similarly, we compute Gaussian curvature as

$$\kappa_{\mathrm{g}} \approx \frac{t_1 t_2 - t_3^2}{t_4^2 + \sqrt{\varepsilon}}.$$

Solving the Linear System

Combining the discretizations calculated above, we obtain the equation:

$$\begin{aligned} m^n_{(i+1/2)jk} u^{n+1}_{(i+1)jk} + m^n_{i(j+1/2)k} u^{n+1}_{i(j+1)k} + m^n_{ij(k+1/2)} u^{n+1}_{ij(k+1)} \\ + m^n_{(i-1/2)jk} u^{n+1}_{(i-1)jk} + m^n_{i(j-1/2)k} u^{n+1}_{i(j-1)k} + m^n_{ij(k-1/2)} u^{n+1}_{ij(k-1)} \\ - \big(m^n_{(i+1/2)jk} + m^n_{i(j+1/2)k} + m^n_{ij(k+1/2)} \\ + m^n_{(i-1/2)jk} + m^n_{i(j-1/2)k} + m^n_{ij(k-1/2)} - \lambda_{\Omega\setminus D} \big) u^{n+1}_{ijk} = r(u^n_{ijk}). \end{aligned} \tag{4.15}$$

We enforce the Dirichlet boundary condition by replacing Equation (4.15) with

$$u^{n+1}_{ijk} = u^n_{ijk} \tag{4.16}$$

for all i, j, k on the boundary of u^n. The discretized coefficients and right-hand side have a $5 \times 5 \times 5$ stencil, so the boundary of u^n is two pixels thick on all sides. For an image u^n of width N, height M, and depth P, taking $i = 1, \ldots, N$, $j = 1, \ldots, M$, and $k = 1, \ldots, P$ gives a sparse system of NMP linear equations:

$$A(u^n)\vec{u}^{\,n+1} = \vec{r}(u^n).$$

For $\vec{u}^{\,n+1}$ and $\vec{r}(u^n)$, we construct 1D vectors in column, row, page order using the

Parameter	**Value**
Inner iterations after which `gmres` restarts	30
Stopping tolerance	10^{-6}
Maximum total `gmres` iterations	500
M1 preconditioner	Lower part of ILU(0) of $A(u^n)$
M2 preconditioner	Upper part of ILU(0) of $A(u^n)$

Table 4.5: `gmres` parameters

element index in Matlab (e.g. `u(:)`). This gives vectors of elements arranged as:

$$\vec{u}^{\,n+1} = \begin{bmatrix} u_{1,1,1}^{n+1} \\ u_{1,2,1}^{n+1} \\ \vdots \\ u_{2,1,1}^{n+1} \\ u_{2,2,1}^{n+1} \\ \vdots \\ u_{1,1,2}^{n+1} \\ u_{1,2,2}^{n+1} \\ \vdots \end{bmatrix}.$$

The second-order implicitly solved derivative operator on the left-hand side has a seven point $3 \times 3 \times 3$ stencil, so the $NMP \times NMP$ coefficient matrix $A(u^n)$ has up to seven non-zero elements in each row. These correspond to the coefficients of elements of u^{n+1} in the left-hand side of Equations (4.15) and (4.16).

To initialize the fixed point iteration, we set

$$u_{ijk}^0 = 0.5 \qquad \text{for } (i, j, k) \text{ in } D.$$

We solve the linear system using the preconditioned generalized minimum residual method [41] (`gmres` in Matlab). The parameters we use for `gmres` are given in Table 4.5. The parameters for the fixed point iteration and mean/Gaussian model are given in Table 4.6.

Parameter	**Value**
Δx	1
ε	10^{-8}
λ	100
C_1	350

Table 4.6: Parameters for 3D mean/Gauss elastica fixed point method

4.4.4 Numerical Results

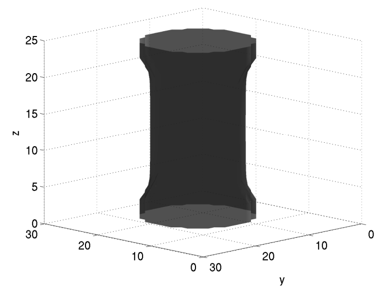
Figures 4.7(a), 4.7(b), and 4.7(c) compare results for the mean and mean/Gaussian curvature models on the cylinder test case. Notice that the model recovers a cylindrical shape as desired when the Gaussian curvature term is used. Figures 4.7(d), 4.7(e), and 4.7(f) give the results of applying the mean and mean/Gaussian curvature models to the box test case. Both models recover a shape that curves inward, but the shape recovered by the mean/Gaussian model does so more gradually than that recovered by the mean curvature model. It is difficult to see in the figure, but taking a slice of the volume at $z \approx 12.5$ reveals that the middle of the recovered object has a circular shape. This could be a result of the mean curvature term as Gaussian curvature is zero on the entire surface of the box. However, this result is desirable for cell path inpainting, since the model should prefer cylindrical shapes. Finally, the tube case in Figures 4.7(g), 4.7(h), and 4.7(i) shows a promising result: the mean/Gaussian curvature model reconnects the tube over a large gap when the mean curvature model does not. This suggests that the mean/Gaussian curvature models has a stronger tolerance to wide gaps and to shapes that do not line up across D. Again, this is highly desirable for cell path inpainting. Figures 4.7(j), 4.7(k), and 4.7(l) show the results for the concentric cylinders case. Neither model recovers the desired output for this case. When the mean and Gaussian curvature terms are weighted equally, the outer cylinder is recovered. When either term is weighted more heavily, the first inner cylinder is recovered. This looks like a disconnected cylinder in the figures because the first inner cylinder has value 0.5 and we show the 0.99 level surface. The hourglass shape in Figures 4.7(s) and 4.7(t) illustrate the preference of both models for a convex shape: the hourglass shape bulges out near its middle in both cases. In the remaining results, the

Test case	a	b	c	C_1
Cylinder	0	20	0	$C_1 < 100$
Tube	0	20	0	$C_1 < 100$
Hourglass	0	20	0	$C_1 < 100$
Cylinder	0	10	10	$C_1 < 60$
Tube	0	10	10	$C_1 < 60$
Hourglass	0	10	10	$C_1 < 60$
Cylinder	0	20	125	$C_1 < 180, C_1 = 240, C_1 = 280, C_1 = 400$
Tube	0	20	125	$C_1 < 240, C_1 = 340$
Hourglass	0	20	125	$C_1 < 260, 280 < C_1 < 340, C_1 = 360, 420 < C_1 < 480$

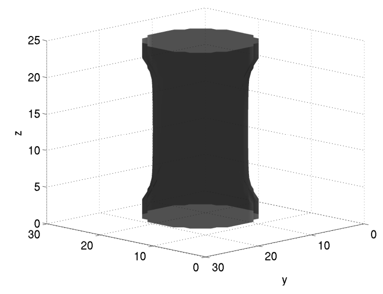
Table 4.7: Values of C_1 in $0, 20, \ldots, 500$ for which the scheme is unstable

mean and mean/Gaussian curvature models recover similar shapes.

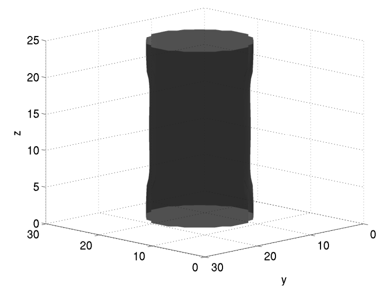
These results are promising, but, when using the Gauss curvature term, it can be difficult to find values of the C_1 parameter for which the numerical scheme is stable. This is illustrated in Table 4.7 where C_1 is varied for fixed a, b, c, u^0 and we report values of C_1 for which the scheme is unstable. For $(a, b, c) = (0, 20, 0)$ and $(a, b, c) = (0, 10, 10)$, the scheme behaves as predicted by Theorem 1: it is stable for sufficiently large values of C_1. Additionally, C_1 seems less sensitive to u^0 in these cases. For $(a, b, c) = (0, 20, 125)$, C_1 is sensitive to u^0. Furthermore, the numerical scheme is slow to converge. The results in Figure 4.7 require approximately 10,000 fixed point iterations.



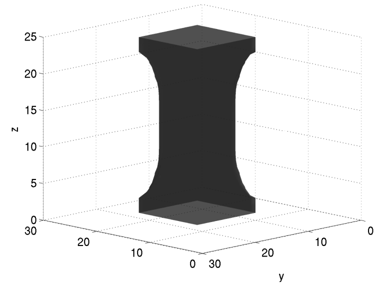
(a) Mean curvature (a=0, b=20, c=0)



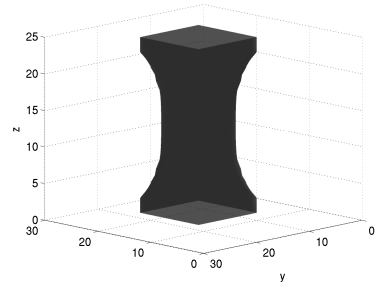
(b) Mean/Gaussian curvature (a=0, b=10, c=10)



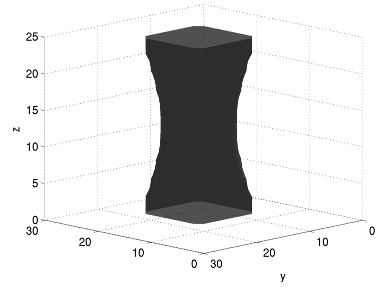
(c) Mean/Gaussian curvature (a=0, b=10, c=125)



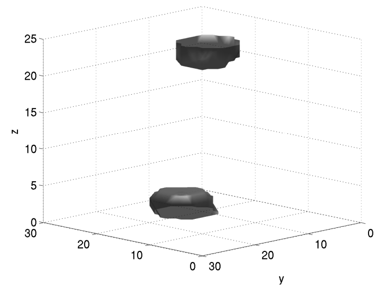
(d) Mean curvature (a=0, b=20, c=0)



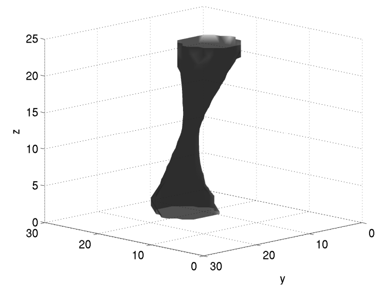
(e) Mean/Gaussian curvature (a=0, b=c=10)



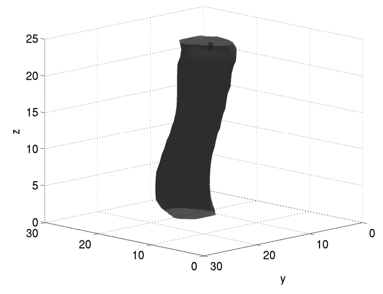
(f) Mean/Gaussian curvature (a=0, b=10, c=125)



(g) Mean curvature (a=0, b=20, c=0)

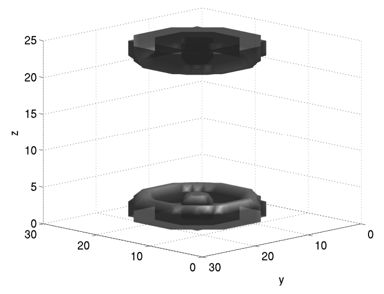


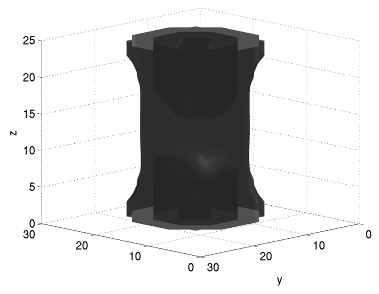
(h) Mean/Gaussian curvature (a=0, b=c=10)

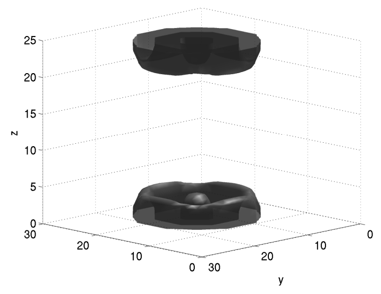


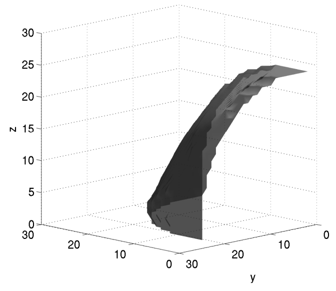
(i) Mean/Gaussian curvature (a=0, b=10, c=125)

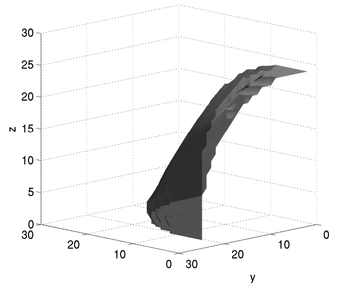
Figure 4.7: 0.99 level surface for elastica models on 3D test cases

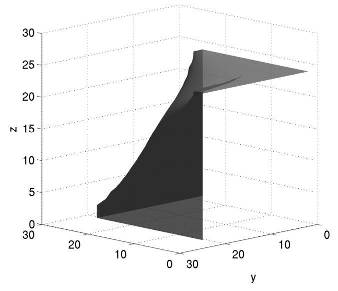

(j) Mean curvature (a=0, b=20, c=0)

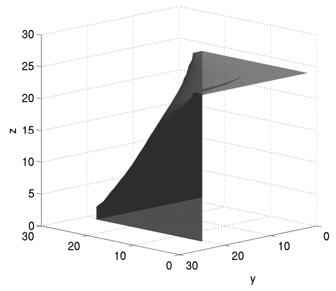

(k) Mean/Gaussian curvature (a=0, b=c=10)

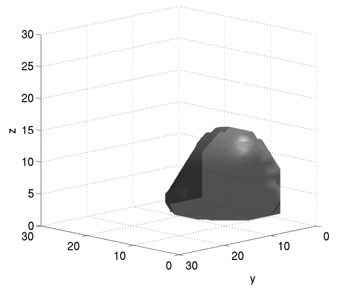

(l) Mean/Gaussian curvature (a=0, b=10, c=125)


(m) Mean curvature (a=0, b=20, c=0)


(n) Mean/Gaussian curvature (a=0, b=c=10)


(o) Mean curvature (a=0, b=20, c=0)


(p) Mean/Gaussian curvature (a=0, b=c=10)


(q) Mean curvature (a=0, b=20, c=0)

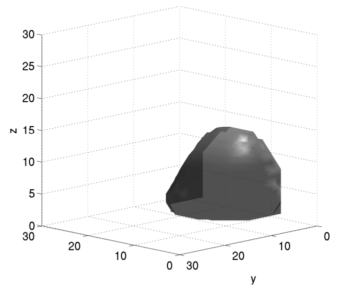
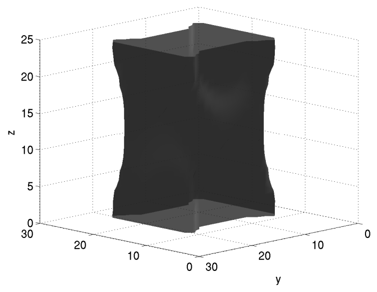
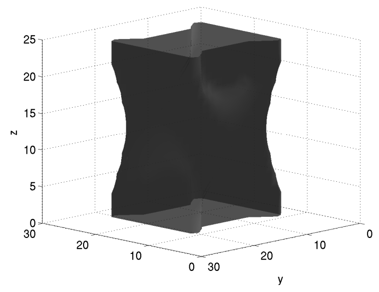

(r) Mean/Gaussian curvature (a=0, b=c=10)

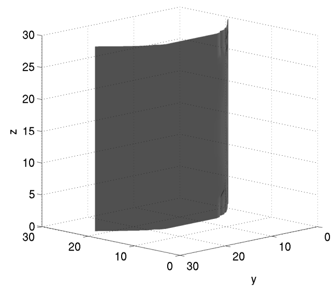
Figure 4.7: 0.99 level surface for elastica models on 3D test cases



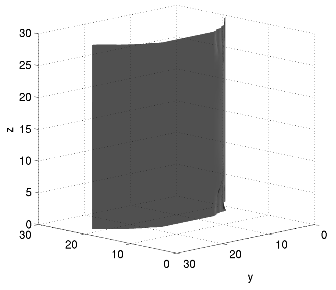
(s) Mean curvature (a=0, b=20, c=0)



(t) Mean/Gaussian curvature (a=0, b=10, c=124)



(u) Mean curvature (a=0.01, b=1, c=0)



(v) Mean/Gaussian curvature (a=0.01, b=1, c=12)

Figure 4.7: 0.99 level surface for elastica models on 3D test cases

Chapter 5

Conclusions and Future Work

2D inpainting methods are well-studied and many have been thoroughly treated analytically. 3D inpainting methods are less well-studied, but have useful applications such as the cell path inpainting problem described in this thesis. A 3D inpainting model based on Gaussian curvature is theoretically appealing and the results presented in this thesis show great promise for cell path reconstruction using the mean/Gaussian elastica model. However, the model is difficult to solve numerically. In our experiments, direct minimization of the mean/Gaussian elastica energy functional is unsuccessful and a fixed-point method for solving the Euler-Lagrange equation is slow to evolve. Numerical methods for solving the mean/Gaussian curvature model present a promising avenue for future research.

References

[1] Uri M. Ascher. *Numerical methods for evolutionary differential equations*, volume 5. SIAM, 2008.

[2] Gilles Aubert and Pierre Kornprobst. *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*. Springer, 2006.

[3] M. Bertalmio, A.L. Bertozzi, and G. Sapiro. Navier-stokes, fluid dynamics, and image and video inpainting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 355–362, 2001.

[4] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 417–424. ACM P./Addison-Wesley Publishing Co., 2000.

[5] Marcelo Bertalmio, Luminita Vese, Guillermo Sapiro, and Stanley Osher. Simultaneous structure and texture image inpainting. *IEEE Transactions on Image Processing*, 12:882–889, 2003.

[6] A. Bertozzi, Selim Esedoḡlu, and A. Gillette. Analysis of a two-scale cahn-hilliard model for binary image inpainting. *Multiscale Modeling and Simulation*, 6:913–936, 2007.

[7] A.L. Bertozzi, S. Esedoḡlu, and A. Gillette. Inpainting of binary images using the cahn-hilliard equation. *Transactions on Image Processing*, 16:285–291, 2007.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[9] Mary Ann Branch, Thomas F Coleman, and Yuying Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21:1–23, 1999.

[10] Carlos Brito-Loeza and Ke Chen. Fast numerical algorithms for Euler's elastica inpainting model. *International Journal of Modern Math*, 5:157–182, 2010.

[11] Richard H Byrd, Jean Charles Gilbert, and Jorge Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89:149–185, 2000.

[12] Richard H Byrd, Mary E Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9:877–900, 1999.

[13] Richard H Byrd, Robert B Schnabel, and Gerald A Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Mathematical programming*, 40:247–263, 1988.

[14] Frédéric Cao, Yann Gousseau, Simon Masnou, and Patrick Pérez. Geometrically guided exemplar-based inpainting. Submitted. Available at `http://hal.archives-ouvertes.fr/hal-00380394/` Retrieved Sept. 25, 2012, 2009.

[15] Jianhong Chan, Tony F.; Shen. Nontexture inpainting by curvature-driven diffusions. *Journal of Visual Communication and Image Representation*, 12:436–449, 2001.

[16] T. Chan, S. Esedoglu, F. Park, and A. Yip. Total variation image restoration: Overview and recent developments. In Nikos Paragios, Yunmei Chen, and Olivier Faugeras, editors, *Handbook of Mathematical Models in Computer Vision*, pages 17–31. Springer, 2006.

[17] Tony F. Chan, Sung Ha Kang, and Jianhong Shen. Euler's elastica and curvature based inpaintings. *SIAM Journal on Applied Mathematics*, 63:564–592, 2002.

[18] Tony F. Chan and Jianhong Shen. Mathematical models for local nontexture inpaintings. *SIAM Journal on Applied Mathematics*, 62:1019–1043, 2002.

[19] U. Clarenz, U. Diewald, G. Dziuk, M. Rumpf, and R. Rusu. A finite element method for surface restoration with smooth boundary conditions. *Computer Aided Geometric Design*, 21:427–445, 2004.

[20] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13:1200–1212, 2004.

[21] Bin Dong, Aichi Chien, Zuowei Shen, and Stanley Osher. A new multiscale representation for shapes and its application to blood vessel recovery. *SIAM Journal on Scientific Computing*, 32:1724–1739, 2010.

[22] Selim Esedoglu and Jianhong Shen. Digital inpainting based on the mumford-shah-euler image model. *European Journal on Applied Mathematics*, 13:353–370, 2002.

[23] D.J. Eyre. An unconditionally stable one-step scheme for gradient systems. *Unpublished*, 1997. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.313`. Retrieved Jan. 20, 2013.

[24] Philip E Gill, Walter Murray, Michael A Saunders, and Margaret H Wright. Procedures for optimization problems with a mixture of bounds and general linear constraints. *ACM Transactions on Mathematical Software (TOMS)*, 10:282–298, 1984.

[25] Enrico Giusti. *Minimal Surfaces and Functions of Bounded Variation*. Birkhäuser, 1984.

[26] Alfred Gray. *Modern differential geometry of curves and surfaces with Mathematica*. CRC P., 1998.

[27] ShihPing Han. A globally convergent method for nonlinear programming. *Journal of optimization theory and applications*, 22:297–309, 1977.

[28] James Hays and Alexei A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics*, 26, 2007.

[29] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 229–238, 1995.

[30] Horman Igehy and Lucas Pereira. Image replacement through texture synthesis. In *Proceedings of the International Conference on Image Processing*, volume 3, pages 186–189, 1997.

[31] Antonio Marquina and Stanley Osher. Explicit algorithms for a new time dependent model based on level set motion for nonlinear deblurring and noise removal. *SIAM Journal on Scientific Computing*, 22:387–405, 2000.

[32] Simon Masnou. Disocclusion: a variational approach using level lines. *IEEE Transactions on Image Processing*, 11:68–76, 2002.

[33] Simon Masnou and Jean-Michel Morel. Level lines based disocclusion. In *Proceedings of the 1998 International Conference on Image Processing*, volume 3, pages 259–263, oct 1998.

[34] Mathworks. Matlab optimization toolbox: Choosing a solver. `http://www.mathworks.com/help/optim/ug/choosing-a-solver.html`. Retrieved April 15, 2013.

[35] Mathworks. Matlab optimization toolbox: Constrained nonlinear optimization algorithms. `http://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html`. Retrieved April 15, 2013.

[36] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42:577–685, 1989.

[37] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer, 2 edition, 2006.

[38] Ivars Peterson. Filling in blanks: Automating the restoration of a picture's missing pieces. *ScienceNews*, 161:299–300, 2011.

[39] Michael JD Powell. A fast algorithm for nonlinearly constrained optimization calculations. In *Numerical analysis*, pages 144–157. Springer, 1978.

[40] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear phenomena*, 60:259–268, 1992.

[41] Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7:856–869, 1986.

[42] C. Schönlieb and A. Bertozzi. Unconditionally stable schemes for higher order inpainting. *Communications in Mathematical Sciences*, 9(2):413–457, 2011.

[43] Kenneth R. Spring and Michael W. Davidson. Nikon MicroscopyU: Introduction to fluorescence microscopy. `http://www.microscopyu.com/articles/fluorescence/fluorescenceintro.html`. Retrieved Jan 19, 2013.

[44] Jean E. Taylor and John W. Cahn. Linking anisotropic sharp and diffuse surface motion laws via gradient flows. *Journal of Statistical Physics*, 77:183–197, 1994.

[45] Richard A Waltz, José Luis Morales, Jorge Nocedal, and Dominique Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107:391–408, 2006.

[46] Robert Weinstock. *Calculus of Variations With Applications to Physics and Engineering.* Dover, 1974.

[47] Eric W. Weisstein. Curvature. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/Curvature.html`, 2013. Retrieved Jan. 20, 2013.

[48] Eric W. Weisstein. Gaussian curvature. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/GaussianCurvature.html`, 2013. Retrieved Jan. 20, 2013.

[49] Eric W. Weisstein. Minimal surface of revolution. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/MinimalSurfaceofRevolution.html`, 2013. Retrieved Jan. 20, 2013.

[50] Alan L. Yuille, Anand Rangarajan, and A. L. Yuille. The concave-convex procedure (CCCP). *Advances in Neural Information Processing Systems*, 2:1033–1040, 2002.