ABSTRACT

Title of dissertation: A VARIATIONAL SHAPE OPTIMIZATION FRAMEWORK FOR IMAGE SEGMENTATION

Günay Doğan
Doctor of Philosophy, 2006

Dissertation directed by: Professor Ricardo H. Nochetto
Department of Mathematics

Image segmentation is one of the fundamental problems in image processing. The goal is to partition a given image into regions that are uniform with respect to some image features and possibly to extract the region boundaries. Recently methods based on PDEs have been found to be an effective way to address this problem. These methods in general fall under the category of shape optimization, as the typical approach is to assign an energy to a shape, say a curve in 2d, and to deform the curve in a way that decreases its energy. In the end when the optimization terminates, the curve is not only at a minimum of the energy, but also at a boundary in the image.

In this thesis, we emphasize the shape optimization view of image segmentation and develop appropriate tools to pursue the optimization in 2d and 3d. We first review the classes of shape energies that are used within the context of image segmentation. Then we introduce the analytical results that will help us design energy-decreasing deformations or *flows* for given shapes.

We describe the gradient flows minimizing the energies, by taking into account the shape derivative information. In particular we emphasize the flexibility to accommodate different velocity spaces, which we later demonstrate to be quite beneficial. We turn the problem into the solution of a system of linear PDEs on the shape. We describe the corresponding space discretization based on the finite element method and an appropriate time discretization scheme as well.

To handle mesh deterioration and possibly singularities due to motion of nodes, we describe time step control, mesh smoothing and angle width control procedures, also a topology surgery procedure that allows curves to undergo topological changes such as merging and splitting. In addition we introduce space adaptivity algorithms that help maintain accuracy of the method and reduce computational cost as well.

Finally we apply our method to two major shape energies used for image segmentation: the minimal surface model (a.k.a geodesic active contours) and the Mumford-Shah model. For both we demonstrate the effectiveness of our method with several examples.

A VARIATIONAL SHAPE OPTIMIZATION FRAMEWORK FOR IMAGE SEGMENTATION

by

Gunay Dogan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:
Professor Ricardo H. Nochetto, Chair/Advisor
Professor Yiannis Aloimonos
Professor Georg Dolzmann
Professor Howard C. Elman
Professor Jian-Guo Liu

ACKNOWLEDGMENTS

The path leading to this thesis has been long and uneven, and it would not be possible for me to carry out this journey without the help and support of many great people. First and foremost is my advisor, Ricardo H. Nochetto. My debt to him cannot be paid. Through years he has been a constant source of support. He has been my mentor, colleague, friend and much more. He is a remarkable person and it has been my privilege to work with him.

The initial ideas and questions for this thesis materialized during the numerics reading meetings. These were an excellent medium to explore new ideas and I would like to thank Prof. Georg Dolzmann, Prof. John Osborn, Prof. Tobias von Petersdorff for nourishing a stimulating atmosphere. Then these ideas evolved through long hours at my office, where I was accompanied by wonderful people, Khamron and Chensong, also Zehra, Ashwin, whom I would like to thank. My days were enlivened by their company, and by Onur, Enver, Franklin, Ganesh, Shawn, Prashant. Thanks for regularly checking if I am alive in my office.

The work that led to the third chapter of this thesis was done in collaboration with Pedro Morin and Marco Verani. This has been crucial for some other parts of my thesis as well. They are two of the nicest people in the world and it has been a pleasure for me to work with them. I would also like to thank Alfred Schmidt and Eberhard Baensch for valuable feedback and suggestions on the fourth chapter.

Finally I would like to thank Prof. Yiannis Aloimonos, Prof. Georg Dolzmann, Prof. Howard Elman and Prof. Jian-Guo Liu for serving on my committee and sparing their invaluable time to review my thesis and provide feedback.

My biggest debt for being where I am now is to my family, my parents through many long years and my wife Rezarta, for their unconditional support and for believing in me. My family and Rezarta's family have always been with us at difficult times. I am especially grateful to our mothers (or angels) Kevser and Kina, for their help when we needed it most. Also special thanks to Ergys as the person we could always depend on.

If there is one that I should be thankful in my life, that is my wife, Rezarta. She is one that brought warmth and color into my life. Without her love and care, I would be adrift. She gave the greatest gift of my life, our daughter Ilayda. They brighten my day. Even at the end of my longest day, I can go home and feel it is a good day. I am one fortunate person and I am thankful for that.

Here I would like to thank several unnamed friends outside my math world as well, for their kindness and support, for enriching my life. Finally I gratefully acknowledge NSF support that I benefited from at various stages of my thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

Shape optimization problems are ubiquitous in science, engineering and industrial applications. They can be formulated as minimization problems with respect to the shape of a domain Ω in $\mathbb{R}^d$. If $y(\Omega)$ is the solution of the following equation in Ω or on $\partial\Omega$

$$Ly(\Omega) = 0, \tag{1.1}$$

where L is a differential or integral operator, and $J(\Omega, y(\Omega))$ is a cost functional or *energy*, then we consider the minimization problem

$$\Omega^* \in \mathcal{U}_{ad} : \qquad J(\Omega^*, y(\Omega^*)) = \inf_{\Omega \in \mathcal{U}_{ad}} J(\Omega, y(\Omega)), \tag{1.2}$$

where $\mathcal{U}_{ad}$ is a set of admissible domains in $\mathbb{R}^d$. If the problem is purely geometric, namely there is no state constraint (1.1), then we simply denote the functional $J(\Omega)$. The main goal of this thesis is to derive a variational method which explicitly and clearly leads first to design a sequence $\{\Omega_n\}_{n=0}^{\infty}$, starting from an initial configuration Ω_0 to a relative minimum Ω_∞, that decreases the function $n \mapsto J(\Omega_n, y(\Omega_n))$, and next to discretize in time and space, thereby obtaining a natural descent direction. In the end we will demonstrate the method with two examples from image processing, in this way, devise a novel technique to effectively address these problems.

The optimization problem cast in this way may strike us as unusual at first, as the variable in the minimization is not a point in a finite dimensional space,

but a shape, which is infinite-dimensional, and the space for such objects is not obvious. Still problems of this type are quite common in engineering. A typical example would be the optimal design of the aerodynamic profile for a body in a fluid stream [46]. Several other examples from engineering disciplines can be found in [2, 9, 58, 69, 52].

It turns out that this framework is the right abstraction for a number of natural phenomena as well. Crystal growth [65, 66], soap bubbles [5], shapes of biological vesicles [37] are some examples, where nature seems to have chosen to obtain the equilibrium shapes through the minimization of some geometric functionals. These phenomena have been studied in detail. In particular extensive work has been done on the mathematics of these processes in material science. We refer to [4].

Recently image processing has emerged as another area spawning many interesting applications of shape optimization. These have mostly been related to image segmentation, where we try to identify objects or object boundaries in given images. Often one can formulate the segmentation problem in such a way that the sought objects in a given image correspond to the minima of a specified shape energy. We take a closer look at these models in §1.1. The shape optimization models used for image segmentation are the main focus of this thesis.

Shape optimization is a multifarious subject with many sides to it, both at theoretical levels and practical levels. Hence it has attracted the attention of several analysts, computational mathematicians and physical scientists. In particular, on the mathematical side, it is connected to many classic areas, such as calculus of variations, differential geometry, geometric measure theory and free boundary prob-

lems. On one hand it is natural to link shape optimization to calculus of variations as both study optima in infinite-dimensional spaces. Nonetheless they are different in that typically in calculus of variations, the variable is a function as opposed to, say, a curve or surface in shape optimization.

Given a shape optimization problem, there is a number of relevant questions, which we need to comment on:

- Well-posedness of the problem, i.e. existence and uniqueness of an optimal shape;
- Derivation of a flow $\Omega_{n+1} = T\Omega_n$ to obtain a sequence of shapes $\{\Omega_n\}_{n=0}^{\infty}$ that converge to the optimal shape;
- Representation of the shapes;
- Numerical procedures to compute the sequence of shapes $\{\Omega_n\}_{n=0}^{\infty}$ minimizing the given energy.

Existence and uniqueness of optimal shapes. Unlike calculus of variations, the results on this aspect of shape optimization are very limited. Although some work exists addressing this question under specific circumstances [14, 17, 22], the issues are technically challenging, and, in fact, for several innocent choices of shape energies, an optimal shape does not exist. We refer to the monograph by Bucur and Buttazzo [13] as a comprehensive review of the available results. Let us also quote the following excerpt from [13] conveying the essence of the problem:

"It must be noticed that the class $\mathcal{U}_{ad}$ of admissible domains does not

have any linear or convex structure, so in shape optimization problems it is meaningless to speak of convex functionals and similar notions. Moreover, even if several topologies on families of domains are available, in general there is not an a priori choice of a topology in order to apply the direct methods of the calculus of variations, for obtaining the existence of at least an optimal domain.

We want to stress that, as it also happens in other kinds of optimal control problems, in several situations an optimal domain does not exist; this is mainly due to the fact that in many cases the minimizing sequences are highly oscillating and converge to limit objects only in the *relaxed* sense. Then we may have, in these cases, only the existence of a *relaxed solution* that in general is not a domain, and whose characterization may change from problem to problem."

"However, the existence of an optimal domain in the following cases:
i) when severe geometrical constraints on the class of admissible domains are imposed (see §5.1 of [13]);
ii) when the cost functional fulfills some particular qualitative assumptions (see §5.4 of [13]);
iii) when the problem is of a very special type, involving only the eigenvalues of the Laplace operator, when neither geometrical constraints nor monotonicity of the cost are required (see §5.6 of [13])."

Shape sensitivity analysis. This addresses the question of finding energy-

decreasing deformations of the shape. The core of shape sensitivity analysis is perturbing the shape and examining how the shape energy changes under the perturbation. We can then use this information to select a specific deformation that will decrease the shape energy. There has been a substantial amount of work on this aspect of shape optimization in the last three decades and now there exists a practical set of analytical tools that can be applied to large classes of problems. For these we refer to the books [25, 34, 47, 57, 64]. We review some of this material in §2 and demonstrate their application on a number of generic shape energies that are relevant for image segmentation.

Representation of the shapes. This is one of the critical components of shape optimization problems. It has implications both on the analytical side and the computational side. Traditionally parametric shape representation has been a choice, widely used [41, 47, 33]. The shape is defined as a mapping from a $d-1$ dimensional domain to a d-dimensional domain. This has been a very popular representation in image processing as well [44, 48]. It is desirable for its computational efficiency and relative ease of use. Recently the level set method, introduced by Osher and Sethian ([54]), has gained significant popularity in shape optimization. In this case, we represent the shape as the zero level set of a higher dimensional function. One major advantage of this approach is its ability to handle topological changes, such as merging and splitting, automatically. We refer to [53, 61] as general level set references, to [15, 16] for elaborating the use of the level set method in shape optimization context. Also we refer to [1, 35, 36, 55, 69] for some applications of this approach. A diffuse interface representation is another possibility that falls

into the Eulerian category like the level set method. The main idea is to represent the shape with a diffuse characteristic function. We refer to [24] for an overview of these representations specifically within the context of geometric flows.

In this thesis we pursue a Lagrangian approach and represent the shapes as polygons in 2d and polyhedra in 3d. In this sense our choice of representation is close to parametric methods. But it is different in that we do not have an explicit parametrization. For the 2d case, we also introduce special procedures in Chapter 4 to add the capability to handle topological changes.

Numerical implementation. The key issues for the numerical implementation are the space discretization and the time discretization. The space discretization is naturally tied to the shape representation. In particular, it depends on what we have chosen as the primary target of our optimization scheme; the domain or the boundary, and how we decide to track it, for example, embedded in an implicit representation such as a level set function or given by a set of polygons or polyhedra. Furthermore we may need to solve integral or differential equations in the domain or on the boundary as well. Then we need to make a choice on how we discretize the equation. This involves numerical quadrature as well as standard techniques such as finite elements, finite difference, spectral methods, etc. Related issues are accuracy of the scheme, computational efficiency, stability and convergence. The choices to make are very much dependent on the problem at hand. We refer to [41], [33], [47], [15], [16] for some practical examples and details of the corresponding scenarios.

For this work, where image segmentation is the main emphasis, we have chosen to track the boundary, which is a polygon in 2d or a polyhedron in 3d. Whenever

we need the domain for specific computations (in 2d), we generate a triangulation that approximates the domain and use that to pursue our computations. We use the finite element method to solve the PDEs that arise on these during the optimization (details in Chapter 3). Additionally we introduce appropriate space adaptivity procedures in Chapter 4 to better leverage accuracy and computational efficiency.

Time discretization is the other critical aspect. One is often inclined to liken the shape optimization problems to geometric evolution problems. However this analogy is not accurate as the time step in shape optimization is artificial. In fact it is more appropriate to view it as a step size in the descent direction. Then the goal is to reach the optimum in as few time steps as possible. A crucial point is to enforce energy decrease at each step. This is a classic problem in nonlinear optimization and procedures such as line search and backtracking are widely used to address this in shape optimization. We mention these in Chapter 3.

In the rest of the introduction we briefly describe our basic model problems and the notion of gradient flow. They are widely used models of image segmentation with distinct behavior and requirements, which can still be studied within a unified framework. We make also explicit the concept of shape derivative of $J(\Omega)$ in the direction of a normal velocity V, namely

$$dJ(\Omega; V) = \int_{\Gamma} GV dS, \tag{1.3}$$

but derive the expressions of G in §2.2 for each case. We then indicate how to exploit this information to design a gradient flow. Note even at this point that if we have an explicit representation of G available, then the choice of velocity $V = -G$ would

set $dJ(\Omega; V) \leqslant 0$ and give us an energy-decreasing flow. Throughout the thesis we will denote with Γ that part of the boundary of Ω which is free to deform, with κ the sum of the principal curvatures of Γ and with ν the unit outer normal of Γ; thus $V := \vec{V} \cdot \nu$. We use the sign convention that a circle with outward normal has positive mean curvature. The symbol $\langle \cdot, \cdot \rangle$ stands for either the L^2-scalar product or a duality pairing on Γ.

1.1 Shape Optimization Problems in Image Processing

In image processing we see applications of shape optimization within the context of image segmentation, where the goal is to partition the image into regions that are uniform with respect to some image features. For example, in medical imaging one might seek regions of uniform intensity representing different tissues. In computer vision, the goal could be segmentation of natural images with respect to texture or possibly more sophisticated features. As often as the regions or the image objects are the targets, one is equally interested in also capturing the region boundaries. An example where the boundaries are the primary target is 3D medical image segmentation, to visualize the biological structures (say organs) in the images. Apart from its significance on its own, image segmentation is also critical for several more sophisticated image processing tasks, for example, image understanding or surgery planning; segmentation is their first step. The literature on image segmentation is overwhelming and it is not possible to review it within the scope of this work. We will instead revisit those that have shape optimization as their main

theme.

Many approaches have been taken to address this problem of image segmentation, building upon various mathematical tools. A relatively recent approach that has gained popularity is based on partial differential equations (PDEs). PDE-based methods have been employed successfully, specifically within the context of boundary extraction. Traditionally edge detector filters are used to distinguish object boundaries. However these serve more as boundary indicators and do not really provide a boundary representation. The PDE approach to this problem has been to use curve evolution (or surface evolution in 3D) techniques to capture the boundaries. Its main advantage is that it directly provides a representation as its outcome.

The first work that takes the curve evolution perspective is the Kass, Witkin, Terzopoulos model [38], also known as the snakes model. In their work Kass et. al. start with a parametric curve close to the sought region boundary and evolve it with respect to some forces computed from image features and some regularity constraints. The curves essentially try to minimize the following energy:

$$J(\Gamma) = \int_a^b |\Gamma'(p)|^2 dp + c_1 \int_a^b |\Gamma''(p)|^2 dp + c_2 \int_a^b h(|\nabla I(\Gamma(p))|) dp, \quad c_1, c_2 > 0, \ (1.4)$$

where $\Gamma : [a, b] \rightarrow \mathcal{U}$, $\mathcal{U}$ is the 2d image domain; $I(x)$ is the image intensity function. The first two terms in (1.4) constitute the internal energy imposing regularity of the curve. The last term is the external energy, where $h(|\nabla I(x)|)$ is a suitable function to enable attraction of the curve to region boundaries. These are signalled by high gradients of image intensity.

Although (1.4) is the first step in the image processing community towards

shape optimization, it is not a truly geometric optimization problem. The reason for this is that the energy (1.4) depends on the parametrization of the curve. For different parametrizations we may get different energy values for the same shape. Still (1.4) has been a very popular method, especially in medical image processing, and many extensions and variations (including parametric surfaces) have been proposed. See [44, 48, 63] for an extensive review of the parametric approach to image segmentation. The first truly geometric shape optimization model is the geodesic active contour model of Caselles, Kimmel and Sapiro [18]. We review it in the next section.

1.1.1 The Minimal Surface Models

Caselles, Kimmel and Sapiro proposed the following energy for curves in [18]

$$J(\Gamma) = \int_a^b h(|\nabla I(\Gamma(p))|)|\Gamma'(p)|dp \tag{1.5}$$

where $h(s) := \frac{1}{1+s^2/\lambda^2}, \quad \lambda > 0$. The function $h(|\nabla I(x)|)$ serves as an edge indicator function and λ is an edge contrast parameter. In Figure 1.1 we give a simple image example and the corresponding edge indicator function. Equation (1.5) is essentially a weighted length formula and the curve that minimizes this energy is the one with minimal length given by this formula. The minima often coincide with an object boundary in the given image. Caselles et al. proposed the 3d version of this model in [19]. They apply the same idea as weighted surfaces to capture boundaries in 3d volume images. An important advantage of the energy (1.5) is that it is intrinsic, i.e. it does not depend on the parametrization of the curve. Also the idea easily

translates to any number of dimensions. For this reason, we consider the following more general version of the energy

$$J(\Gamma) = \int_{\Gamma} H(x)dS + \gamma \int_{\Omega} H(x)dx \tag{1.6}$$

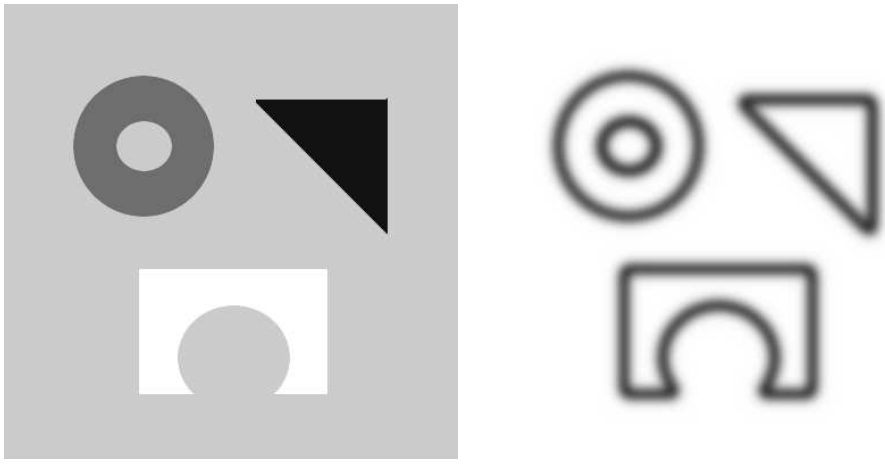
where Γ is a closed $d-1$ dimensional surface in d-dimensional space, Ω is the volume enclosed by Γ, and dS denotes the surface measure; $H(x) > 0$ is generic weight function, possibly equal to $h(|\nabla I(x)|)$. The second term is often added to speed up the computation and to help detection of concavities. Let us remark that models similar to (1.6) had already been studied extensively (see for example [4], [24] and the references therein) before its introduction to the image processing community. We will see that the explicit form of the shape gradient of (1.6) is given by

$$G = H(x)\kappa + \partial_{\nu} H(x) + \gamma H(x)$$

and one can use this to derive an energy-decreasing deformation for the surface Γ for example given by the following velocity

$$\vec{V} = -G\nu = -\left(H(x)\kappa + \partial_{\nu} H(x) + \gamma H(x)\right)\nu.$$

In [39], Kimmel and Bruckstein propose an anisotropic version of the active contour model. They point out that the geodesic contour model does not perform well on images with varying image gradients on the background, in particular, it has difficulty detecting object concavities. To improve behavior in these situations, they introduce an edge alignment term $\langle \nabla I, \nu \rangle$ that will push the curve to align its normal with the image gradient to achieve better positioning of the curve. The



$I(x)$ $h(|\nabla I(x)|)$

Figure 1.1: A synthetic image example and the corresponding edge indicator function. The dark areas in the right figure indicate small values of $h(|\nabla I(x)|)$.

generic form of the energy is as follows

$$J(\Gamma) := \int_{\Gamma} H(x, \nu) dS. \tag{1.7}$$

Keriven and Faugeras work with another model of the same form in [30], where they try to reconstruct a 3d scene from given 2d camera images. The energy (1.7) has been studied in other contexts too [66]. The explicit form of the shape gradient G for this energy is

$$G = H(x, \nu)\kappa + \partial_{\nu} H(x, \nu) + \mathrm{div}_{\Gamma}(H_y(x, \nu))_{\Gamma}$$

where $H_y(\cdot, \cdot)$ is the gradient with respect to the second vector variable, i.e. the normal ν, div_{Γ} is the tangential divergence and $(\vec{\omega})_{\Gamma}$ denotes the restriction of the vector $\vec{\omega}$ to the tangent plane of the surface Γ.

The energy (1.7) is distinct from (1.6) in that the optimization not only depends on a given external weight function, but also on the geometry of the shape. Another possible interaction with the geometry could be through its mean curvature,

κ. The generic form of such a functional is

$$J(\Gamma) := \int_{\Gamma} H(x, \kappa) dS. \tag{1.8}$$

Leventon et al. consider a functional of this form in [42], where they try to force the curves to conform to some prescribed curvature within the context of medical image segmentation. Note that $H(x, \kappa) = \kappa^2$ gives the well-known Willmore functional (see [68]) and as another possibility, one can use this as an intrinsic regularization term with other functionals. The explicit form of the shape gradient G for this energy is

$$G = -\Delta_{\Gamma} \left(H_z(x, \kappa) \right) + H(x, \kappa)\kappa - H_z(x, \kappa) \sum_i \kappa_i^2 + \partial_{\nu} H(x, \kappa)$$

where $H_z(\cdot, \cdot)$ is the derivative with respect to the second variable, i.e. the mean curvature κ, Δ_{Γ} is the Laplace-Beltrami operator or the tangential Laplacian and κ_i are the principal curvatures.

1.1.2 Models with Integrals and Statistics

The models we mentioned in §1.1.1 have one significant drawback in practice. As they are minimal length formulas (for curves in 2D), the algorithms based on these may fail to increase the length of the curves where it is necessary to get a complete segmentation. For example this may happen when we try to detect concavities of objects in images. This difficulty manifests itself often in medical imaging when the target is thin and long structures in the image. A possible approach to circumvent this issue is to consider an average of the edge indicator over the curve,

rather than its integral. Then the energy has the following form

$$J(\Gamma) = \frac{1}{|\Gamma|} \int_{\Gamma} H(x) dS.$$

This is proposed in [31] by Fua and Leclerc. Following this, Desolneux et al. argue in [27] that one should instead optimize the average of an edge alignment term, which results in an energy of the form

$$J(\Gamma) = \frac{1}{|\Gamma|} \int_{\Gamma} H(x, \nu) dS.$$

As we define $|\Gamma| := \int_{\Gamma} dS$, these models have now modified the energies (1.6), (1.7) by adding integral quantities. These create additional dependency of the energy on the shape Γ as the integrals themselves also depend on the shape.

Integrals in shape energies are adopted for implementation of more sophisticated segmentation ideas too. For example, in segmentation of natural images, one often seeks to incorporate the statistics of the region inside the curve and the region outside the curve. In general these statistics are computed as integrals over the regions. Examples of these can be found in [21, 8, 56, 59]. A simple example of statistics that can be used for this purpose is the mean value of the image intensity in the regions. Chan and Vese propose a method based on this idea in [21]. They minimize the following energy

$$J(\Omega, \mu) = \sum_{i=1,2} \int_{\Omega_i} (I(x) - \mu_i)^2 + \gamma \int_{\Gamma} dS$$

where

$$\mu = \mu_1 \chi_{\Omega_1} + \mu_2 \chi_{\Omega_2}, \qquad \mu_i = \int_{\Omega_i} I(x) dx, \quad i = 1, 2,$$

Ω_1 denotes the region inside the curve Γ and $\Omega_2 = D - \Omega_1$ denotes the region outside Γ. This method is particularly suited for two-phase images, i.e. images that can be decomposed into a background and a foreground, each with roughly constant intensity.

Rather than examining each of these separately, we will consider two forms that we think are general representatives of shape energies with integrals. The first is a boundary energy:

$$J(\Gamma, I_w(\Gamma)) = \int_\Gamma H(x, I_w(\Gamma))dS, \quad I_w(\Gamma) = \int_\Gamma w(x)dS. \tag{1.9}$$

The second one is a domain energy:

$$J(\Omega, I_w(\Omega)) = \int_\Omega H(x, I_w(\Omega))dx, \quad I_w(\Omega) = \int_\Omega w(x)dx. \tag{1.10}$$

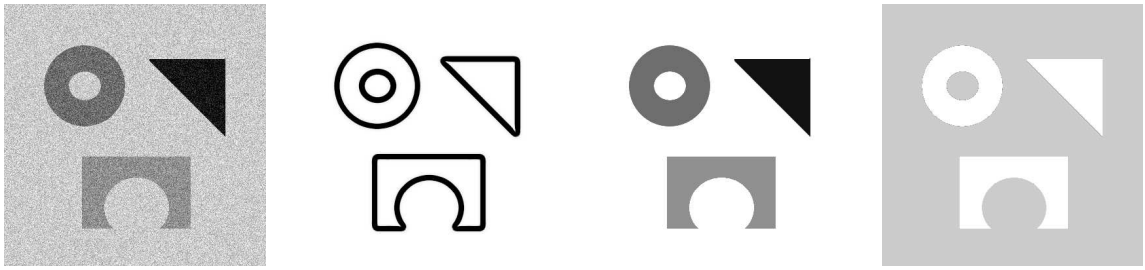
We could certainly add more integral quantities to these energies as $I^1_{w_1}(\Omega)$, $I^2_{w_2}(\Omega)$, ..., $I^n_{w_n}(\Omega)$ or add higher level of dependence on the domain, such as having $w = w(x, \Omega)$ in $I_w(\Omega)$. These are discussed in some detail in [8]. However, for our purposes we do not see that this additional complexity brings any additional insight, so we stick to the compact versions, (1.9) and (1.10). We will see the shape gradients for these energies are given by

$$G = \big(H(x, I_w(\Gamma)) + I_{H_p}(\Gamma)w(x)\big)\,\kappa + \partial_\nu H(x, I_w(\Gamma)) + I_{H_p}(\Omega)\partial_\nu w(x)$$

and

$$G = H(x, I_w(\Omega)) + I_{H_p}(\Gamma)w(x)$$

for the boundary and domain energies respectively; H_p denotes the derivative of H with respect to its integral variable.



$I(x)$ discontinuities foreground background

Figure 1.2: Using the Mumford-Shah model to perform simultaneous segmentation and denoising on a synthetic image. From left to right, the figures show the original noisy image, the set of discontinuities obtained, the denoised foreground, the denoised background.

1.1.3 The Mumford-Shah Model

In addition to integrals, we can use PDEs to estimate quantities in the regions. This is exemplified by the work of Chan and Vese [20], and Tsai et al. [67]. These two are very similar and both give a curve evolution solution for the Mumford-Shah model, which we review briefly. The Mumford-Shah model was proposed in [49] by Mumford and Shah. The goal is to find a set of discontinuities K and a smooth approximation u to the image. This is to be achieved by the following optimization problem

$$\min_{u,K} \left\{ \frac{1}{2} \int_D (u - I)^2 dx + \frac{\mu}{2} \int_{D-K} |\nabla u|^2 dx + \gamma \, length(K) \right\}. \tag{1.11}$$

The problem in this form is hard as two variables u, K of the optimization problem are of very different nature. So one looks at a modified version of this energy. One approach is to look for a diffuse set of discontinuities and we replace K with the a function that represents the diffuse discontinuities. This idea was proposed in [6] by Ambrosio and Tortorelli. An adaptive finite element solution for their model is described in [12]. This approach addresses more of the image restoration aspect of

the Mumford-Shah model. Still note that the original Mumford-Shah model in fact aims at both image restoration and image segmentation.

Chan and Vese relate (1.11) to previous segmentation work based on active contours. They constrain the discontinuity set to be a set of closed curves and in this way they turn it into a traditional shape optimization problem. The Chan-Vese approach to the Mumford-Shah problem consists of minimizing

$$J(\Gamma) = \sum_{i=1}^{2} \frac{1}{2} \left(\int_{\Omega_i} (u_i - I)^2 + \mu |\nabla u_i|^2 \right) dx + \gamma \int_{\Gamma} dS. \tag{1.12}$$

The Euler-Lagrange equation for Γ fixed reduces to

$$\begin{cases} -\mu \Delta u_i + u_i &= I \quad \text{in} \quad \Omega_i \\ \partial_{\nu_i} u_i &= 0 \quad \text{on} \quad \partial\Omega_i \end{cases}$$

with $i = 1, 2$ where the curve(s) Γ partitions the domain into a foreground Ω_1 (inside the curve) and a background Ω_2 (outside the curve). The explicit form of G for this energy is

$$G = \frac{1}{2} [\![|u - I|^2]\!] + \frac{\mu}{2} [\![|\nabla u|^2]\!] + \gamma\kappa,$$

with $[\![f]\!] = f_1 - f_2$ indicating the jump of f across Γ.

1.2 Thesis Outline and Contributions

Let us first observe that in all the examples above, the function G in (1.3) has the form

$$G = g(x, \Omega)\kappa + f(x, \Omega) \tag{1.13}$$

except for the case with curvature dependence in the energy.

The explicit expression for G can be exploited to deform Ω in the direction V of maximal decrease of the functional $J(\Omega, y(\Omega))$. To do this, we first introduce a bilinear form $b(\cdot,\cdot)$ on Γ which induces a scalar product, and next consider the gradient flow

$$b(V,W) = -\int_\Gamma GW, \qquad \forall W, \tag{1.14}$$

where Γ (and hence G) implicitly depend on $\vec{V} = V\nu$ by means of a suitable system of ODE describing the deformation of Ω through V. If $\mathcal{B}$ is a (elliptic) operator such that $\langle \mathcal{B}V, W\rangle = b(V,W)$, then (1.14) is equivalent to solving the elliptic PDE on the surface Γ for the normal velocity V

$$\mathcal{B}V = -G. \tag{1.15}$$

We point out that so far we have not discretized the underlying problem but still have been able to find a descent direction for the domain shape, the *steepest descent* direction. The next step is to discretize in time explicitly or alternatively semi-implicitly in order to retain the implicit computation of curvature in (1.13), for stability purposes, but not the full geometry. This time discretization is fully discussed in §3.2 and is followed by space discretization via finite element methods in §3.3. The ensuing variational approach is rather flexible to accommodate several scalar products $b(\cdot,\cdot)$ depending on the application, as discussed in §3.2.2 and §5. This flexibility will prove to be beneficial as we will be able to compare traditional choices of gradient flows in image processing with possible variations and we will document the superiority of the latter.

The outline of the thesis is as follows:

- In *Chapter 1*, we give a brief introduction to shape optimization and discuss its relevance in image processing within the context of the image segmentation problem. We describe the classes of shape energies used for this purpose.

- In *Chapter 2*, we review the basic tools of shape sensitivity analysis and demonstrate how they apply to the shape energies we discuss in Chapter 1. We survey existing shape calculus results for these energies and derive new results to complement existing results, including first and second shape derivatives of the energies.

- In *Chapter 3*, we describe the time and space discretization for energy-decreasing flows. For this we represent the energy descent direction as the solution of a system of linear PDEs and describe the notion of explicit and semi-implicit time discretization within this system. We follow with the space discretization based on the finite element method and we explain how to solve the resulting linear system.

- In *Chapter 4*, we introduce a number of computational enhancements that help maintain the quality of computations and to increase the effectiveness of the method in terms of accuracy and computational efficiency. For mesh quality, we introduce time step control, mesh smoothing and angle width control for surfaces. To improve effectiveness we describe geometry-driven and data-driven space adaptivity procedures, and also a topology surgery procedure that provides the capability to undergo topological changes such as merging and splitting.

- In *Chapter 5*, we consider two major models used for image segmentation: the minimal surface model (a.k.a geodesic active contour model) and the Mumford-Shah model. For these we put together the material of the previous chapters and describe in detail how we address these models. We document the effectiveness of the method with several experiments.

Our specific contributions are as follows:

- In *Chapter 2*, we derive the first shape derivatives of the general curvature dependent boundary energy (1.8) and the boundary energy with integrals (1.9) and the second shape derivatives of the anisotropic boundary energy (1.7), the boundary energy with integrals (1.9) and the domain energy with integrals (1.10). Moreover, this is the first work that provides an comprehensive listing of the generic forms of shape energies used in image processing and gives the corresponding shape sensitivity analysis.

- In *Chapter 3*, the time and space discretization for the gradient flows, albeit inspired by [29, 11, 15], are completely novel. This was produced in collaboration with P. Morin, R. H. Nochetto and M. Verani. An important feature of the discretization is the flexibility to use different scalar products to constrain the space of deformation velocities. In particular, one can introduce scalar products addressing the specific needs of the given application. See [28] for examples.

- In *Chapter 4*, we introduce space adaptivity to improve accuracy and efficiency. The main ideas are not original. But their implementation in the way described

is completely novel for the problem of image segmentation. To our knowledge, there do not exist such procedures to address the shape functionals in image segmentation. Moreover the algorithm for topological changes in 2d is novel. There exists other work that include topological changes in 2d. But their treatment of the problem is relatively superficial. None tackle the problem in the generality that we do. In particular discussion of possible pitfalls and pathological cases is largely omitted. Our algorithm, on the other hand, has been tested with numerous experiments and simulations and has proven to be a robust and reliable method.

- In *Chapter 5*, we give novel methods to address the minimal surface model (1.6) and the Mumford-Shah model (1.12). Curve evolution approaches for these models are based on the level set method. Our methods are radically different from existing work in that we track the geometric objects explicitly based on the discretization described in Chapter 3. In particular, the incorporation of the adaptivity procedures in Chapter 4, frees the method from the image pixels and allows us to pursue the optimization with much fewer computational elements than the image resolution would dictate. We demonstrate the methods with several synthetic and real examples.

Chapter 2

Shape Sensitivity Analysis

In §2.1 we introduce some elements of shape calculus, along with related references, necessary to properly carry out the shape sensitivity analysis of the model problems in §1.1.

2.1 Shape Differential Calculus

We start by briefly recalling some useful notions of differential geometry. Let us be given $h \in C^2(\Gamma)$ and an extension $\tilde{h}$ of h, $\tilde{h} \in C^2(U)$ and $\tilde{h}|_\Gamma = h$ on Γ where U is a tubular neighborhood of Γ in $\mathbb{R}^d$. Then the *tangential gradient* $\nabla_\Gamma h$ of h is defined as follows:

$$\nabla_\Gamma h = \big(\nabla \tilde{h} - \partial_\nu \tilde{h}\ \nu\big)|_\Gamma,$$

where ν denotes the unit normal vector to Γ. For $\vec{W} \in [C^2(\Gamma)]^d$ properly extended to a neighborhood of Γ, we define the *tangential divergence* of $\vec{W}$ by

$$\mathrm{div}_\Gamma \vec{W} = \big(\mathrm{div}\vec{W} - \nu \cdot D\vec{W} \cdot \nu\big)|_\Gamma, \tag{2.1}$$

where $D\vec{W}$ denotes the Jacobian matrix of $\vec{W}$. Finally, if $D^2\tilde{h}$ denotes the Hessian of $\tilde{h}$, then the *Laplace-Beltrami operator* Δ_Γ on Γ is defined as follows:

$$\Delta_\Gamma h = \mathrm{div}_\Gamma(\nabla_\Gamma h) = \big(\Delta \tilde{h} - \nu \cdot D^2\tilde{h} \cdot \nu - \kappa\ \partial_\nu \tilde{h}\big)|_\Gamma. \tag{2.2}$$

2.1.1 The Velocity Method

We consider now a hold-all domain $\mathcal{D}$, which contains Ω, and a vector field $\vec{V}$ defined on $\mathcal{D}$, which is used to define the continuous sequence of perturbed sets $\{\Omega_t\}_{t\geq 0}$, with $\Omega_0 := \Omega$. Each point $x \in \Omega_0$ is continuously deformed by an ODE defined by the field $\vec{V}$. The parameter which controls the amplitude of the deformation is denoted by t (a fictitious time).

We now consider the system of autonomous ODEs

$$\frac{dx}{dt} = \vec{V}(x(t)), \quad \forall t \in [0,T], \qquad x(0) = X, \tag{2.3}$$

where $X \in \Omega_0 = \Omega$. This defines the mapping

$$x(t,\cdot) : X \in \Omega \to x(t,X) \in \mathbb{R}^d, \tag{2.4}$$

and also the perturbed sets

$$\Omega_t = \{x(t,X) : \quad X \in \Omega_0\}. \tag{2.5}$$

We recall that the family of perturbed sets has its regularity preserved for $\vec{V}$ smooth enough [64]: if Ω_0 is of class C^r, $r \leq k$, then for any $t \in [0,T]$, Ω_t is also of class C^r.

2.1.2 Derivative of Shape Functionals

Let $J(\Omega)$ be a shape functional; examples of such functionals have been given in §1.1. The Eulerian derivative, or *shape derivative*, of the functional $J(\Omega)$ at Ω, in the direction of the vector field $\vec{V}$ is defined as the limit

$$dJ(\Omega;\vec{V}) = \lim_{t\to 0} \frac{1}{t}\big(J(\Omega_t) - J(\Omega)\big). \tag{2.6}$$

Let B be a Hilbert space of perturbating vector fields. The functional $J(\Omega)$ is said to be shape differentiable at Ω in B if the shape derivative $dJ(\Omega;\vec{V})$ exists for all $\vec{V} \in B$ and the mapping $\vec{V} \to dJ(\Omega;\vec{V})$ is linear and continuous on B. An analogous definition can be introduced for functionals $J(\Gamma)$ depending on a $d-1$ manifold Γ as an independent variable.

We now recall a series of results from shape differential calculus in $\mathbb{R}^d$. We start with the shape derivative of domain and boundary integrals of functions not depending on the geometry.

Lemma 2.1.1 ([64, Prop.2.45]). *Let $\phi \in W^{1,1}(\mathbb{R}^d)$ and $\Omega \subset \mathbb{R}^d$ be open and bounded. Then the functional*

$$J(\Omega) = \int_\Omega \phi dx \tag{2.7}$$

is shape differentiable. The shape derivative of J is given by

$$dJ(\Omega;\vec{V}) = \int_\Omega \operatorname{div}(\phi\vec{V})dx. \tag{2.8}$$

If $\Gamma = \partial\Omega$ is of class C^1 and $V = \vec{V}\cdot\nu$, then

$$dJ(\Omega;\vec{V}) = \int_\Gamma \phi V dS. \tag{2.9}$$

Proof. Let $T_t(X) = x(X,t)$ be the mapping defined by (2.3) and (2.4). We want to compute

$$dJ(\Omega;V) = \frac{d}{dt}J(\Omega_t)|_{t=0}$$

where $\Omega_t := T_t(\Omega)$ and

$$J(\Omega_t) = \int_{\Omega_t} \phi dx = \int_\Omega \phi \circ T_t J_t dX.$$

The Jacobian is given by $J_t := \det DT_t(X)$. We compute the following intermediate results omitting the details:

$$\begin{aligned} \frac{d}{dt}\phi \circ T_t &= (\nabla\phi \cdot \vec{V}) \circ T_t, \\ \frac{dJ_t}{dt} &= ((\mathrm{div}\vec{V}) \circ T_t)J_t. \end{aligned}$$

Now we can compute the first shape derivative of the domain functional

$$\begin{aligned} dJ(\Omega; \vec{V}) &= \int_\Omega \left(\frac{d}{dt}(\phi \circ T_t)J_t + \phi \circ T_t \frac{dJ_t}{dt} \right) dx|_{t=0} \\ &= \int_\Omega \left(\nabla\phi \cdot \vec{V} + \phi \mathrm{div}\vec{V} \right) dx \\ &= \int_\Omega \mathrm{div}(\phi\vec{V}) dx \\ &= \int_\Gamma \phi\vec{V} \cdot \nu dS. \end{aligned}$$

□

Lemma 2.1.2 ([64, Prop. 2.50 and (2.145)]). *Let $\psi \in W^{2,1}(\mathbb{R}^d)$ and Γ be of class C^2. Then the functional*

$$J(\Gamma) = \int_\Gamma \psi dS \tag{2.10}$$

is shape differentiable and

$$dJ(\Gamma; \vec{V}) = \int_\Gamma \big(\nabla\psi \cdot \vec{V} + \psi \mathrm{div}_\Gamma \vec{V} \big) dS = \int_\Gamma \big(\partial_\nu \psi + \psi\kappa \big) V dS. \tag{2.11}$$

Proof. As above let $T_t(X) = x(X,t)$ be the mapping defined by (2.3) and (2.4). Now we want to compute

$$dJ(\Gamma; V) = \frac{d}{dt} J(\Gamma_t)|_{t=0}$$

where $\Gamma_t := T_t(\Gamma)$ and

$$J(\Gamma_t) = \int_{\Gamma_t} \psi dS_t = \int_\Gamma \psi \circ T_t \omega_t dS.$$

The change of variables brings $\omega_t := J_t|^* DT_t^{-1}\nu|$, where we denote the transpose of DT by ${}^*DT$. We compute the following intermediate results omitting the details:

$$\begin{aligned} \frac{d}{dt}\psi \circ T_t &= (\nabla\psi \cdot \vec{V}) \circ T_t, \\ \frac{dJ_t}{dt} &= \mathrm{div}\vec{V} - \nu \cdot D\vec{V} \cdot \nu. \end{aligned}$$

Now we can compute the first shape derivative of the boundary functional

$$\begin{aligned} dJ(\psi; \vec{V}) &= \int_\Gamma \left(\frac{d}{dt}(\psi \circ T_t)\omega_t + \psi \circ T_t \frac{d\omega_t}{dt} \right) dS|_{t=0} && (2.12) \\ &= \int_\Gamma \Big(\nabla\psi \cdot \nu + \psi(\mathrm{div}\vec{V} - \nu \cdot D\vec{V} \cdot \nu) \Big) dS && (2.13) \\ &= \int_\Gamma (\partial_\nu \psi + \kappa\psi)\, \vec{V} \cdot \nu dS. && (2.14) \end{aligned}$$

The last line follows from Proposition 2.1.1. □

Let us now consider more general functionals $J(\Omega)$. These are useful when we consider some of the problems in §1.1.1, 1.1.2, 1.1.3. In particular we are interested in computing sensitivities for functionals of the form

$$J(\Omega) = \int_\Omega \phi(x, \Omega) dx, \qquad \text{or} \qquad J(\Gamma) = \int_\Gamma \varphi(x, \Gamma) dS, \tag{2.15}$$

where the functions $\phi(\cdot, \Omega) : \Omega \to \mathbb{R}$ and $\varphi(\cdot, \Gamma) : \Gamma \to \mathbb{R}$ themselves depend on the geometric variables Ω and Γ, respectively. To handle the computation of the sensitivities of such functionals we need to take care of the derivatives of ϕ and φ with respect to Ω and Γ.

First of all we recall the notions of *material derivative* and *shape derivative*.

Definition 2.1.1 ([64, Prop.2.71]). *The material derivative $\dot{\phi}(\Omega;\vec{V})$ of ϕ at Ω in direction $\vec{V}$ is defined as follows*

$$\dot{\phi}(\Omega;\vec{V}) = \lim_{t\to 0}\frac{1}{t}\big(\phi(x(t,\cdot),\Omega_t) - \phi(\cdot,\Omega_0)\big), \tag{2.16}$$

where the mapping $x(\cdot,t)$ is defined as in (2.4). A similar definition holds for functions $\varphi(\cdot,\Gamma)$ which are defined on boundaries Γ instead of domains Ω.

Definition 2.1.2 ([64, Def. 2.85, Def 2.88]). *The shape derivative $\phi'(\Omega;\vec{V})$ of ϕ at Ω in the direction $\vec{V}$ is defined to be*

$$\phi'(\Omega;\vec{V}) = \dot{\phi}(\Omega;\vec{V}) - \nabla\phi\cdot\vec{V}. \tag{2.17}$$

Accordingly, for boundary functions $\varphi(\Gamma):\Gamma\to\mathbb{R}$, the shape derivative is defined to be

$$\varphi'(\Gamma;\vec{V}) = \dot{\varphi}(\Gamma;\vec{V}) - \nabla_\Gamma\varphi\cdot\vec{V}|_\Gamma. \tag{2.18}$$

With these notions we are able to calculate the shape derivatives for the above shape functionals.

Theorem 2.1.1 ([64, Sect. 2.31, 2.33]). *Let $\phi=\phi(x,\Omega)$ be given so that the material derivative $\dot{\phi}(\Omega;\vec{V})$ and the shape derivative $\phi'(\Omega;\vec{V})$ exist. Then, the cost functional $J(\Omega)$ in (2.15) is shape differentiable and we have*

$$dJ(\Omega;\vec{V}) = \int_\Omega \phi'(\Omega;\vec{V})dx + \int_\Gamma \phi V dS. \tag{2.19}$$

For boundary functions $\varphi(\Gamma)$, the shape derivative of $J(\Gamma)$ in (2.15) is given by

$$dJ(\Gamma;\vec{V}) = \int_\Gamma \varphi'(\Gamma;\vec{V})dS + \int_\Gamma \kappa\varphi V dS, \tag{2.20}$$

whereas if $\varphi(\cdot,\Gamma)=\psi(\cdot,\Omega)|_\Gamma$*, then we obtain*

$$dJ(\Gamma;\vec{V})=\int_\Gamma \psi'(\Omega;\vec{V})|_\Gamma dS+\int_\Gamma \big(\partial_\nu\psi+\kappa\psi\big)V dS. \tag{2.21}$$

Let us conclude this part with a Riesz representation theorem, the Hadamard-Zolésio Theorem, that will play an important role in the sequel.

Theorem 2.1.2 ([64, Sect 2.11 and Th. 2.27]). *The shape derivative of a domain or boundary functional always has a representation of the form*

$$dJ(\Omega;\vec{V})=\langle G,V\rangle_\Gamma, \tag{2.22}$$

where we denote by $\langle\cdot,\cdot\rangle_\Gamma$ *a suitable duality pairing on* Γ*; that is, the shape derivative is concentrated on* Γ*.*

Let us point out that an implication of this theorem is that the shape derivative $dJ(\Omega,\vec{V})$ depends only on $V=\vec{V}\cdot\nu$, the normal component of the velocity. *For this reason, we will use* V *in our notation from now on and assume a normal extension when we need the velocity extended to a neighborhood of the surface. Hence, without loss of generality, we have the following assumptions on* V *and* $\vec{V}$

$$\vec{V}=V\nu, \qquad \partial_\nu V=0 \quad \text{ on } \Gamma. \tag{2.23}$$

2.1.3 Some Geometric Results

In this section we present some results that will be useful to compute the first and second shape derivatives of the model problems.

Lemma 2.1.3. *The shape derivatives of the normal ν and the mean curvature κ of a boundary Γ of class C^2 with respect to velocity V are given by*

$$\nu' = \nu'(\Gamma; V) = -\nabla_\Gamma V, \tag{2.24}$$

$$\kappa' = \kappa'(\Gamma; V) = -\Delta_\Gamma V. \tag{2.25}$$

Then the shape derivatives of $\partial_\nu f$, $\nabla_\Gamma f$ and $|\nabla_\Gamma f|^2$ for a C^1 function $f(x, \Gamma)$ are

$$(\partial_\nu f)' = \partial_\nu f' - \partial f \cdot \nabla_\Gamma V \tag{2.26}$$

$$(\nabla_\Gamma f)' = \nabla_\Gamma f' + \partial_\nu f \, \nabla_\Gamma V + (\nabla_\Gamma f \cdot \nabla_\Gamma V)\nu \tag{2.27}$$

$$(|\nabla_\Gamma f|^2)' = 2\nabla_\Gamma f \cdot \nabla_\Gamma f' + 2\partial_\nu f \, \nabla_\Gamma f \cdot \nabla_\Gamma V \tag{2.28}$$

where $f' = f'(\Gamma; V)$ is the shape derivative of f.

Proof. The main idea of proving the results for ν' and κ' is to consider the signed distance representation $b(x)$ of the boundary Γ. It is defined as

$$b(x) = \begin{cases} \text{dist}(x, \Gamma) & \text{for } x \in \mathbb{R}^d - \Omega \\ 0 & \text{for } x \in \Gamma \\ -\text{dist}(x, \Gamma) & \text{for } x \in \Omega \end{cases} \tag{2.29}$$

where

$$\text{dist}(x, \Gamma) = \inf_{y \in \Gamma} |y - x|.$$

The signed distance representation of Γ allows us to extend ν and κ smoothly in a tubular neighborhood. We use the fact that

$$\nu = \nabla b(x)|_\Gamma, \quad \kappa = \Delta b(x)|_\Gamma, \quad II = D^2 b(x)|_\Gamma$$

where II denotes the second fundamental form (see [25, Sect. 8.5]). Once we have the extensions, it is relatively straight-forward to compute ν' and κ'. For this we use the result [35] that

$$\phi(\cdot,\Omega) \in H^{\frac{3}{2}+\epsilon}(\Omega), \quad \phi(\cdot,\Omega)|_\Gamma = 0,$$

implies existence of $\phi'(\Omega;V)$ in $H^{\frac{1}{2}+\epsilon}(\Omega)$ and

$$\phi'(\Omega;V)|_\Gamma = -\partial_\nu \phi\, V|_\Gamma.$$

Noting $b|_\Gamma = 0$, we get

$$b'|_\Gamma = \partial_\nu b|_\Gamma V = -\nabla b \cdot \nabla b|_\Gamma V = -V.$$

On the other hand

$$0 = (1)' = (\nabla b \cdot \nabla b)' = 2\nabla b' \cdot \nabla b,$$

gives

$$\nu' = \nabla b'|_\Gamma = \nabla_\Gamma b'|_\Gamma + \nabla b'|_\Gamma \cdot \nu\nu = \nabla_\Gamma b'|_\Gamma = -\nabla_\Gamma V.$$

Similarly we compute $\kappa' = \Delta b'|_\Gamma = -\Delta_\Gamma V$, whose derivation we omit.

Now given ν' and κ' we can compute the shape derivatives of the normal and tangential derivatives of f.

$$\begin{aligned}(\partial_\nu f)' &= (\nabla f \cdot \nu)' = \nabla f' \cdot \nu - \nabla f \cdot \nabla_\Gamma V \\ &= \partial_\nu f' - \nabla f \cdot \nabla_\Gamma V.\end{aligned}$$

From this

$$\begin{aligned}(\nabla_\Gamma f)' &= (\nabla f - \partial_\nu f \nu)' \\ &= \nabla f' - (\partial_\nu f)'\nu - \partial_\nu f \nu' \\ &= \nabla f' - \partial_\nu f'\nu + (\nabla f \cdot \nabla_\Gamma V)\nu + \partial_\nu f \nabla_\Gamma V \\ &= \nabla_\Gamma f' + (\nabla f \cdot \nabla_\Gamma V)\nu + \partial_\nu f \nabla_\Gamma V.\end{aligned}$$

Finally

$$\begin{aligned}\left(|\nabla_\Gamma f|^2\right)' &= 2\nabla_\Gamma f \cdot (\nabla_\Gamma f)' \\ &= 2\nabla_\Gamma f \cdot \nabla_\Gamma f' + 2\partial_\nu f \nabla_\Gamma f \cdot \nabla_\Gamma V\end{aligned}$$

since $\nabla_\Gamma f \cdot \nu = 0$. □

Lemma 2.1.4. *The normal derivative of the mean curvature of a surface Γ of class C^2 is given by*

$$\partial_\nu \kappa = -\sum_i \kappa_i^2 \tag{2.30}$$

where κ_i denote the principal curvatures of the surface. For a two-dimensional surface in 3D, this is equal to

$$\partial_\nu \kappa = -(\kappa_1^2 + \kappa_2^2) = -(\kappa^2 - 2\kappa_G)$$

where $\kappa_G = \kappa_1 \kappa_2$ denotes the Gaussian curvature.

Proof. To prove this result, we will work with the signed distance representation

$b(x)$ of Γ (see (2.29)). Referring to (2.30) again, we proceed as follows

$$\begin{aligned} 0 &= \Delta(1) = \Delta(\nabla b \cdot \nabla b) = \partial_{x_i}\partial_{x_i}(\partial_{x_j} b\, \partial_{x_j} b) \\ &= \partial_{x_i}\left(\partial_{x_i x_j} b\, \partial_{x_j} b + \partial_{x_j} b\, \partial_{x_i x_j} b\right) = 2\partial_{x_i}(\partial_{x_j} b\, \partial_{x_i x_j} b) \\ &= 2\left(\partial_{x_i x_j} b\, \partial_{x_i x_j} b + (\partial_{x_j}\partial_{x_i x_i} b)\, \partial_{x_j} b\right) \\ &= 2\left(D^2 b : D^2 b + \nabla(\Delta b) \cdot \nabla b\right). \end{aligned}$$

Then

$$\partial_\nu \kappa = \nabla(\Delta b) \cdot \nabla b|_\Gamma = -\|D^2 b\|_F^2|_\Gamma = -\|II\|_F^2 = -\sum_i \kappa_i^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. To get this result we used the fact that squared Frobenius norm of a square matrix is equal to the sum of the squares of its eigenvalues, also that the eigenvalues of the second fundamental form correspond to the principal curvatures. □

Lemma 2.1.5. *Let $f = f(x, \kappa)$ be given such that $f \in C^2(\mathbb{R}^{d+1})$ and Γ be of class C^4. Then the Laplace-Beltrami operator on f is given by*

$$\Delta_\Gamma\left(f(x,\kappa)\right) = \Delta_\Gamma f + f_z \Delta_\Gamma \kappa + f_{zz}|\nabla_\Gamma \kappa|^2 + 2\nabla_\Gamma f_z \cdot \nabla_\Gamma \kappa$$

where f_z and f_{zz} denote the first and second derivatives of f with respect to its curvature variable.

Proof. Recall the definition of the Laplace-Beltrami operator

$$\Delta_\Gamma f = \Delta f - \nu \cdot D^2 f \cdot \nu - \partial_\nu f \kappa$$

or alternatively

$$\Delta_\Gamma f = \left(\Delta f - \nabla b \cdot D^2 f \cdot \nabla b - \nabla f \cdot \nabla b \Delta b\right)|_\Gamma.$$

Now let us compute each component of these expressions.

$$\begin{aligned}
\partial_{x_j}\left(f(x,\Delta b(x))\right) &= f_{x_j} + f_z\partial_{x_j}\Delta b,\\
\partial_{x_i}\partial_{x_j}\left(f(x,\Delta b(x))\right) &= f_{x_jx_i} + f_{x_jz}\partial_{x_i}\Delta b + f_{zx_i}\partial_{x_j}\Delta b\\
&\quad + f_{zz}\partial_{x_j}\Delta b\partial_{x_i}\Delta b + f_z\partial_{x_i}\partial_{x_j}\Delta b\\
\partial_{x_i}\partial_{x_i}\left(f(x,\Delta b(x))\right) &= \Delta f + (\nabla f)_z\cdot\nabla\Delta b + \nabla f_z\cdot\nabla\Delta b\\
&\quad + f_{zz}(\nabla\Delta b)\cdot(\nabla\Delta b) + f_z\Delta\Delta b.
\end{aligned}$$

Then

$$\begin{aligned}
\partial_{x_i}b\,\partial_{x_i}\partial_{x_j}f\,\partial_{x_j}b &= \nabla b\cdot D^2f\cdot\nabla b + \nabla b\cdot(\nabla f)_z\nabla b\cdot\nabla\Delta b\\
&\quad + \nabla b\cdot(\nabla f_z)\nabla b\cdot\nabla\Delta b + f_{zz}(\nabla b\cdot\nabla\Delta b)^2 + f_z\nabla b\cdot D^2\Delta b\nabla b,\\
\partial_{x_i}f\partial_{x_i}b\Delta b &= \nabla f\cdot\nabla b\Delta b + f_z\nabla b\cdot\nabla\Delta b.
\end{aligned}$$

Now we add these up. Recalling $\nabla_\Gamma h = \left(\nabla h - (\nabla b\cdot\nabla h)\nabla b\right)|_\Gamma$ and reorganizing the terms, we have

$$\begin{aligned}
\Delta_\Gamma f &= \Delta f - \nabla b\cdot D^2f\cdot\nabla b - \nabla f\cdot\nabla b\Delta b\\
&\quad + f_z\left(\Delta\Delta b - \nabla b\cdot D^2\Delta b\cdot\nabla b - \nabla\Delta b\cdot\nabla b\Delta b\right)\\
&\quad + f_{zz}|\nabla_\Gamma\Delta b|^2 + \nabla_\Gamma f_z\cdot\nabla_\Gamma\Delta b + (\nabla_\Gamma f)_z\cdot\nabla_\Gamma\Delta b.
\end{aligned}$$

Restricting this expression to Γ to get $\nu = \nabla b(x)|_\Gamma$ and $\kappa = \Delta b(x)|_\Gamma$ and using the definition (2.2) of the Laplace-Beltrami operator, we obtain

$$\Delta_\Gamma\left(f(x,\kappa)\right) = \Delta_\Gamma f + f_z\Delta_\Gamma\kappa + f_{zz}|\nabla_\Gamma\kappa|^2 + 2\nabla_\Gamma f_z\cdot\nabla_\Gamma\kappa$$

□

Proposition 2.1.1 ([25, Sect. 8.5.5, (5.27)]). *For a function $f \in C^1(\Gamma)$ and a vector $\vec{\omega} \in C^1(\Gamma)^d$, we have the following tangential Green's formula*

$$\int_\Gamma f \mathrm{div}_\Gamma \vec{\omega} + \nabla_\Gamma f \cdot \vec{\omega} dS = \int_\Gamma \kappa f \vec{\omega} \cdot \nu dS. \tag{2.31}$$

2.1.4 The Second Shape Derivative

We continue to use the scalar velocity fields V, W satisfying (2.23) to perturb the domains and we define the second shape derivative as follows

$$d^2 J(\Omega; V, W) = d\left(dJ(\Omega; V)\right)(\Omega; W). \tag{2.32}$$

The second shape derivatives of functions $\phi(\Omega)$, $\varphi(\Gamma)$ can be defined similarly based on Definition 2.1.2. Now we can use this to compute the second shape derivative of the domain and the boundary functionals. We give these results below.

Lemma 2.1.6 ([35, Sect. 2.5]). *Let $\phi \in W^{2,1}(\mathbb{R}^d)$ and Γ be of class C^2. Then the second shape derivative of the functional*

$$J(\Omega) = \int_\Omega \phi dx \tag{2.33}$$

at Ω with respect to scalar velocity fields V, W is given by

$$d^2 J(\Omega; V, W) = \int_\Gamma \left(\partial_\nu \phi + \kappa \phi\right) VW dS. \tag{2.34}$$

Proof. Recall that the first shape derivative of (2.33) is

$$dJ(\Omega; V) = \int_\Gamma \phi V dS. \tag{2.35}$$

If we let $\Phi = \phi V$, then the second shape derivative of (2.33) is given by

$$d^2J(\Omega; V, W) = d(dJ(\Omega; V))(\Omega; W) \tag{2.36}$$

$$= \int_\Gamma \Phi' dS + \int_\Gamma (\partial_\nu \Phi + \Phi\kappa)\, W dS, \tag{2.37}$$

where $\Phi' = \Phi'(\Omega; W)$. As neither of ϕ, V have shape dependence, we have $\Phi' = 0$. Let us compute the second term.

$$\partial_\nu \Phi = \partial_\nu \phi V + \phi \partial_\nu V. \tag{2.38}$$

Note that the second term is zero because of the assumptions (2.23). Substituting Φ and $\partial_\nu \Phi$ in (2.37) gives the result. □

Lemma 2.1.7 ([35, Sect. 5]). *Let $\psi \in W^{3,1}(\mathbb{R}^d)$ and Γ be of class C^2. Then the second shape derivative of the functional*

$$J(\Gamma) = \int_\Gamma \psi dS \tag{2.39}$$

at Γ with respect to scalar velocity fields V, W is given by

$$d^2J(\Gamma; V, W) = \int_\Gamma \left(\psi \nabla_\Gamma V \cdot \nabla_\Gamma W + \left(\partial_{\nu\nu}\psi + 2\kappa \partial_\nu \psi + (\kappa^2 - \sum \kappa_i^2)\psi \right) VW \right) dS.$$

Proof. The first shape derivative of (2.39) is

$$dJ(\Gamma; V) = \int_\Gamma (\partial_\nu \psi + \kappa\psi) V dS. \tag{2.40}$$

If we let $\Psi = (\partial_\nu \psi + \kappa\psi)V$, then the second shape derivative of (2.39) is given by

$$d^2J(\Gamma; V, W) = d(dJ(\Gamma; V))(\Gamma; W) \tag{2.41}$$

$$= \int_\Gamma \Psi' dS + \int_\Gamma (\partial_\nu \Psi + \Psi\kappa)\, W dS. \tag{2.42}$$

where $\Psi' = \Psi(\Gamma; W)$. Let us compute the first term:

$$\begin{aligned} \Psi' &= (\nabla\psi \cdot \nu' + \psi\kappa')\, V \\ &= -(\nabla\psi \cdot \nabla_\Gamma W + \psi\Delta_\Gamma W)\, V, \end{aligned}$$

using Lemma 2.1.3 and the fact that ψ, V do not depend on the shape. Integrate Ψ'

$$\begin{aligned} \int_\Gamma \Psi' dS &= -\int_\Gamma \nabla_\Gamma\psi \cdot \nabla_\Gamma W V dS - \int_\Gamma \psi V \Delta_\Gamma W dS \\ &= \int_\Gamma \psi \nabla_\Gamma V \cdot \nabla_\Gamma W dS, \end{aligned}$$

using the tangential Green's formula (2.31). Now reminding $\partial_\nu V = 0$ by assumption (2.23), we compute

$$\begin{aligned} \partial_\nu \Psi &= \partial_\nu \left(\partial_\nu\psi + \psi\kappa\right) V \\ &= \left(\partial_{\nu\nu}\psi + \kappa\partial_\nu\psi - \psi \sum \kappa_i^2\right) V. \end{aligned}$$

We have used the facts: $\partial_\nu \nu = 0$ and $\partial_\nu \kappa = -\sum \kappa_i^2$ from Lemma 2.1.4. Substituting Ψ' and $\partial_\nu \Psi$ in (2.42), we obtain the explicit expression for the second shape derivative. □

Now we employ Lemmas 2.1.6 and 2.1.7 and Definition 2.1.2 to state the second shape derivative for more general functionals $J(\Omega)$. The assumptions (2.23) are crucial in obtaining this result.

Theorem 2.1.3. *Let $\phi = \phi(x, \Omega)$ be given so that the first and the second shape derivatives $\phi'(\Omega; V)$, $\phi''(\Omega; V, W)$ exist. Then, the second shape derivative of the*

domain functionals in (2.15) is given by

$$d^2J(\Omega;V,W)=\int_\Omega \phi''dx+\int_\Gamma \left(\phi'_W V+\phi'_V W\right)dS+\int_\Gamma \left(\partial_\nu\phi+\kappa\phi\right)VW dS \quad (2.43)$$

where $\phi'_V=\phi'(\Omega;V)$, $\phi''=\phi''(\Omega;V,W)$. *For boundary functions* $\varphi(\Gamma)$ *in (2.15) with* $\varphi(\cdot,\Gamma)=\psi(\cdot,\Omega)|_\Gamma$ *we obtain*

$$\begin{aligned} d^2J(\Gamma;V,W) &= \int_\Gamma \psi''dS+\int_\Gamma \left(\left(\partial_\nu\psi'_W+\kappa\psi'_W\right)V+\left(\partial_\nu\psi'_V+\kappa\psi'_V\right)W\right)dS \quad (2.44)\\ &+ \int_\Gamma \left(\psi\nabla_\Gamma V\cdot\nabla_\Gamma W+\left(\partial_{\nu\nu}\psi+2\kappa\partial_\nu\psi+(\kappa^2-\sum\kappa_i^2)\psi\right)VW\right)dS. \end{aligned}$$

where $\psi'_V=\psi'(\Gamma;V)|_\Gamma$, $\psi''=\psi'(\Gamma;V,W)|_\Gamma$.

Proof. For $J(\Omega)=\int_\Omega\phi(x,\Omega)dx$, the first shape derivative at Ω in direction V is

$$dJ(\Omega;V) = \underbrace{\int_\Omega \phi'(\Omega;V)dx}_{J_1} + \underbrace{\int_\Gamma \phi V dS}_{J_2}.$$

To obtain the second shape derivative, we take the derivatives of J_1 and J_2.

$$\begin{aligned} dJ_1(\Omega;W) &= \int_\Omega \phi''(\Omega;V,W)dx+\int_\Gamma \phi'(\Omega;V)WdS\\ dJ_2(\Omega;W) &= \int_\Gamma \phi'(\Omega;W)VdS+\int_\Gamma(\partial_\nu\phi V+\phi V\kappa)dS. \end{aligned}$$

Then

$$dJ(\Omega;V,W)=dJ_1(\Omega;W)+dJ_2(\Omega;W)$$

and reorganizing the terms yields the results. We obtain the second shape derivative for the boundary functional similarly. □

More details on the structure of the second shape derivative can be found in [25] and [51]. We should nevertheless point out that the use of assumptions (2.23) has allowed us to derive the simpler forms in Theorem 2.1.3 (compared to [25] and [51]).

2.2 Shape Derivatives of the Model Functionals

For each shape functional introduced in §1.1 we compute the first and second shape derivatives and obtain the explicit expressions of the shape gradient G.

2.2.1 Minimal Surface Energies

We compute the first and the second shape derivatives of the general form of the active shape model (1.6)

$$J(\Omega) := \int_\Gamma H(x)dS + \gamma \int_\Omega H(x)dx$$

Theorem 2.2.1. *The first shape derivative of the general form of the active shape model is given by*

$$dJ(\Omega; V) = \int_\Gamma \Big((\kappa + \gamma)H(x) + \partial_\nu H(x)\Big)VdS.$$

Proof. The proof directly follows from Lemmas 2.1.1, 2.1.2. □

Then the explicit form of shape gradient for (1.6) is

$$G = (\kappa + \gamma)H(x) + \partial_\nu H(x),$$

and has the form (1.13) with $g = H(x)$ and $f = \gamma H(x) + \partial_\nu H(x)$.

Theorem 2.2.2. *The second shape derivative of the general form of the active shape model is given by*

$$\begin{aligned} d^2J(\Gamma; V, W) &= \int_\Gamma H\nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &\quad + \int_\Gamma \Big(\partial_{\nu\nu}H + (2\kappa + \gamma)\partial_\nu H + (\kappa^2 - \sum \kappa_i^2 + 2\gamma\kappa)H\Big)VWdS. \end{aligned}$$

Proof. The proof directly follows from Lemmas 2.1.6, 2.1.7. □

Now we compute the first and the second shape derivatives of the anisotropic boundary energy (1.7)

$$J(\Gamma) := \int_\Gamma H(x,\nu)dS$$

Theorem 2.2.3. *The first shape derivative of the anisotropic boundary energy is given by*

$$\begin{aligned} dJ(\Gamma;V) &= \int_\Gamma (\kappa H + \partial_\nu H)\, V - H_y \cdot \nabla_\Gamma V dS \\ &= \int_\Gamma (\kappa H + \partial_\nu H + \mathrm{div}_\Gamma (H_y)_\Gamma)\, V dS. \end{aligned}$$

Proof. We use Theorem 2.1.1 with $\psi = H(x,\nu)$. Then

$$\psi'(\Gamma;V) = H_y \cdot \nu' = -H_y \cdot \nabla_\Gamma V$$

using (2.24); H_y is the derivative with respect to the second variable. Note also that $\partial_\nu(H(x,\nu)) = \partial_\nu H(x,\nu)$ since $\partial_\nu \nu = 0$. Then substituting ψ' in (2.21) yields

$$dJ(\Gamma;V) = \int_\Gamma (\kappa H + \partial_\nu H)\, V - H_y \cdot \nabla_\Gamma V dS$$

On the other hand we can apply the tangential Green's formula (2.31) to the last term of the integral.

$$\int_\Gamma H_y \cdot \nabla_\Gamma V dS = -\int_\Gamma V \mathrm{div}_\Gamma H_y dS + \int_\Gamma \kappa V H_y \cdot \nu dS$$

Since $\mathrm{div}_\Gamma(\vec{\omega})_\Gamma = \mathrm{div}_\Gamma(\vec{\omega}) - \kappa\vec{\omega}\cdot\nu$, we have

$$dJ(\Gamma;V) = \int_\Gamma (\kappa H + \partial_\nu H + \mathrm{div}_\Gamma (H_y)_\Gamma)\, V dS.$$

□

Then the explicit form of the shape gradient for (1.7) is

$$G = \kappa H + \partial_\nu H + \mathrm{div}_\Gamma(H_y)_\Gamma,$$

and has the form (1.13) with $g = H$ and $f = \partial_\nu H + \mathrm{div}_\Gamma(H_y)_\Gamma$.

Theorem 2.2.4. *The second shape derivative of the anisotropic boundary energy is given by*

$$\begin{aligned} d^2J(\Gamma;V,W) &= \int_\Gamma \nabla_\Gamma V \cdot ((H + H_y \cdot \nu)Id + H_{yy}) \cdot \nabla_\Gamma W dS \\ &+ \int_\Gamma \left(\partial_{\nu\nu} H + 2\kappa\partial_\nu H + (\kappa^2 - \sum \kappa_i^2)H\right) VW dS \\ &- \int_\Gamma (\kappa H_y + \nu^T H_{xy}) \cdot (\nabla_\Gamma W\, V + \nabla_\Gamma V\, W) dS. \end{aligned}$$

Proof. We use Theorem 2.1.3 with $\psi = H(x,\nu)$. Then

$$\begin{aligned} \psi'(\Gamma;V) &= -H_y \cdot \nabla_\Gamma V \\ \psi''(\Gamma;V,W) &= -(H_y)' \cdot \nabla_\Gamma V - H_y \cdot (\nabla_\Gamma V)' \\ &= \nabla_\Gamma V \cdot H_{yy} \cdot \nabla_\Gamma W + H_y \cdot \nu\, \nabla_\Gamma V \cdot \nabla_\Gamma W \end{aligned}$$

using (2.27) and (2.23). We collect the terms and obtain

$$\psi''(\Gamma;V,W) = \nabla_\Gamma V \cdot (H_{yy} + H_y \cdot \nu\, Id) \cdot \nabla_\Gamma W$$

where Id is the identity matrix. We also compute

$$\begin{aligned} \partial_\nu \psi'(\Gamma;V) &= -\partial_\nu H_y \cdot \nabla_\Gamma V - H_y \cdot \partial_\nu \nabla_\Gamma V \\ &= -\nu \cdot H_{yx} \cdot \nabla_\Gamma V \end{aligned}$$

since $\partial_\nu \nabla_\Gamma V = \nabla_\Gamma \partial_\nu V = 0$ due to (2.23). The function H_{yx} is the Hessian obtained by taking the derivative of H with respect to the second variable, then with respect

to the first variable. Now we substitute these in (2.44) to obtain

$$\begin{aligned} d^2J(\Gamma;V,W) &= \int_\Gamma \nabla_\Gamma V \cdot (H_{yy} + H_y \cdot \nu\, Id) \cdot \nabla_\Gamma W dS \\ &- \int_\Gamma (\nu \cdot H_{yx} \cdot \nabla_\Gamma W + \kappa H_y \cdot \nabla_\Gamma W)\, V \\ &- \int_\Gamma (\nu \cdot H_{yx} \cdot \nabla_\Gamma V + \kappa H_y \cdot \nabla_\Gamma V)\, W dS \\ &+ \int_\Gamma \left(H \nabla_\Gamma V \cdot \nabla_\Gamma W + \left(\partial_{\nu\nu} H + 2\kappa \partial_\nu H + (\kappa^2 - \sum \kappa_i^2) H \right) VW \right) dS. \end{aligned}$$

Reorganizing the terms yields the result. □

Next we compute the shape derivative of the curvature dependent boundary energy

$$J(\Gamma) := \int_\Gamma H(x,\kappa) dS.$$

Theorem 2.2.5. *The first shape derivative of the curvature dependent boundary energy* (1.8) *is given by*

$$dJ(\Gamma;V) = \int_\Gamma \left(-\Delta_\Gamma (H_z) + H\kappa - H_z \sum \kappa_i^2 + \partial_\nu H \right) V dS,$$

where H_z denotes the derivative with respect to the curvature variable and

$$\Delta_\Gamma (H_z) = \Delta_\Gamma H_z + H_{zz} \Delta_\Gamma \kappa + H_{zzz} |\nabla_\Gamma \kappa|^2 + 2 \nabla_\Gamma H_{zz} \cdot \nabla_\Gamma \kappa.$$

Note the nuance in the notation: $\partial_{x_i} H = \partial_{x_i} H(x,\kappa) = H_{x_i}(x,\kappa)$ where as $\partial_{x_i}(H) = \partial_{x_i}(H(x,\kappa)) = H_{x_i}(x,\kappa) + H_z(x,\kappa)\partial_{x_i}\kappa$.

Proof. We use Theorem 2.1.1 with $\psi = H(x,\kappa)$. Then

$$\psi'(\Gamma;V) = H_z \kappa' = -H_z \Delta_\Gamma V$$

using (2.25). Let us also compute $\partial_\nu(H(x,\kappa))$:

$$\begin{aligned}\partial_\nu\left(H(x,\kappa)\right) &= \partial_\nu H(x,\kappa) + H_z(x,\kappa)\partial_\nu\kappa \\ &= \partial_\nu H(x,\kappa) - H_z(x,\kappa)\sum \kappa_i^2.\end{aligned}$$

The second line follows from (2.30). Then substituting ψ' and $\partial_\nu\psi$ in (2.21) we have

$$dJ(\Gamma;V) = -\int_\Gamma H_z(x,\kappa)\Delta_\Gamma V dS + \int_\Gamma \left(\partial_\nu H(x,\kappa) - H_z(x,\kappa)\sum \kappa_i^2 + H(x,\kappa)\kappa\right) V dS.$$

Integrating the first integral by parts twice with tangential Green's formula (2.31), we obtain

$$dJ(\Gamma;V) = \int_\Gamma \left(-\Delta_\Gamma(H_z(x,\kappa)) + \partial_\nu H(x,\kappa) - H_z(x,\kappa)\sum \kappa_i^2 + H(x,\kappa)\kappa\right) V dS.$$

We can readily compute $\Delta_\Gamma(H_z(x,\kappa))$ using Lemma (2.1.5). □

In this case the explicit form of the shape gradient for (1.8) is

$$G = -\Delta_\Gamma\left(H_z\right) + H\kappa - H_z\sum \kappa_i^2 + \partial_\nu H.$$

Note that this does not comply with the form (1.13). We need to compute the principal curvatures, also derivatives of the curvature.

Corollary 2.2.1. *The Willmore functional*

$$J(\Gamma) = \frac{1}{2}\int_\Gamma \kappa^2 dS$$

admits a first shape derivative $dJ(\Gamma;V) = \int_\Gamma GV$ *with*

$$G = -\Delta_\Gamma\kappa - \kappa\left(\sum \kappa_i^2 - \frac{\kappa^2}{2}\right).$$

For surfaces in 3d, this is equal to

$$G = -\Delta_\Gamma \kappa - \frac{\kappa^3}{2} + 2\kappa_G,$$

where $\kappa_G = \kappa_1 \kappa_2$ is the Gaussian curvature.

Proof. The proof trivially follows from Theorem 2.2.5 by setting $H(x, \kappa) = \frac{\kappa^2}{2}$. Then $\partial_\nu H = 0$ and $H_z = \kappa$. Substituting these values and using (2.30) gives the result. □

2.2.2 Shape Energies with Integrals

We compute the shape derivatives for the boundary energies with integrals (1.9), (1.10). The first shape derivative for (1.10) can be found in [8] as well, where the authors also investigate the dependence of the energy on several integrals.

Theorem 2.2.6. *The first shape derivative of* (1.9)*, namely*

$$J(\Omega) = \int_\Omega H(x, I_w(\Omega))dx, \qquad I_w(\Omega) = \int_\Omega w(x)dx$$

is given by

$$dJ(\Omega; V) = \int_\Gamma \left(H(x, I_w(\Omega)) + I_{H_p}(\Omega) w(x) \right) V dS,$$

where H_p denotes the derivative of $H(x, I_w(\Omega))$ with respect to its second variable.

Proof. Let $\phi(x, \Omega) = H(x, I_w(\Omega))$ in Theorem 2.1.1, and calculate $\phi'(\Omega; V)$.

$$\phi' = H_p(x, I_w) I_w' = H_p \int_\Gamma w(\tilde{x}) V dS.$$

We have used the fact that Lemma 2.1.1 applies to I_w, namely $I_w' = dI_w(\Omega; V)$. Substitute ϕ' in the general form (2.19)

$$dJ(\Omega; V) = \int_\Omega H_p(x, I_w) \left(\int_\Gamma w(\tilde{x}) V dS \right) dx + \int_\Gamma H(x, I_w) V dS.$$

Since $I_{H_p} = \int_\Omega H_p(x, I_w)dx$, exchanging the order of integration we have

$$dJ(\Omega; V) = \int_\Gamma \left(H(x, I_w) + I_{H_p} w(x)\right) V dS.$$

□

Then the explicit form of the shape gradient for (1.10) is

$$G = H(x, I_w) + I_{H_p} w(x),$$

and has the form (1.13) with $g = 0$ and $f = H(x, I_w) + I_{H_p} w(x)$.

Theorem 2.2.7. *The second shape derivative of* (1.10) *is given by*

$$\begin{aligned} dJ(\Omega; V, W) &= \int_\Gamma \left(\partial_\nu H + I_{H_p}\partial_\nu w + \kappa(H + I_{H_p} w)\right) VW dS \\ &+ \int_\Gamma H_p V dS \int_\Gamma wW dS + \int_\Gamma wV dS \int_\Gamma H_p W dS \\ &+ I_{H_{pp}} \int_\Gamma wV dS \int_\Gamma wW dS, \end{aligned}$$

with $H = H(x, I_w(\Omega))$, $w = w(x)$ as above. Functions H_p and H_{pp} denote the first and second derivatives of H with respect to its second variable.

Proof. Again let $\phi = H(x, I_w(\Omega))$ in Theorem 2.1.3. We have

$$\begin{aligned} \phi' &= H_p \int_\Gamma w(\tilde{x}) V dS, \\ \phi'' &= (H_p)' \int_\Gamma w(\tilde{x}) V dS + H_p \left(\int_\Gamma w(\tilde{x}) V dS\right)' \\ &= H_{pp} \int_\Gamma w(\tilde{\tilde{x}}) V dS \int_\Gamma w(\tilde{x}) W dS + H_p \int_\Omega \left(\partial_\nu w(\tilde{x}) + \kappa w(\tilde{x})\right) VW dS, \end{aligned}$$

by Lemma 2.1.6. We substitute these in (2.43)

$$\begin{aligned} d^2 J(\Omega; V, W) &= \int_\Omega H_{pp} dx \int_\Gamma wV dS \int_\Gamma wW dS + \int_\Gamma H_p dx \int_\Omega \left(\partial_\nu w + \kappa w\right) VW dS \\ &+ \int_\Gamma H_p V dS \int_\Gamma wW dS + \int_\Gamma H_p W dS \int_\Gamma wV dS \\ &+ \int_\Gamma (\partial_\nu H + \kappa H) VW dS. \end{aligned}$$

Reorganizing the various terms yields the result. □

Theorem 2.2.8. *The first shape derivative of* (1.9)*, namely*

$$J(\Gamma) = \int_\Gamma H(x, I_w(\Gamma))dS, \qquad I_w(\Gamma) = \int_\Gamma w(x)dS,$$

is given by

$$dJ(\Gamma; V) = \int_\Gamma \left((H + I_{H_p} w)\kappa + \partial_\nu H + I_{H_p}\partial_\nu w\right) V dS,$$

where H_p is the derivative of $H(x, I_w)$ with respect to the second variable.

Proof. Let $\psi(x, \Gamma) = H(x, I_w(\Gamma))$ in Theorem 2.1.1. We calculate $\psi'(\Gamma; V)$ using Lemma 2.1.2.

$$\psi' = H_p(x, I_w)I'_w = H_p \int_\Gamma (\partial_\nu w + w\kappa)V dS$$

Substitute in the general form

$$dJ(\Gamma; V) = \int_\Gamma H_p dS \int_\Gamma (\partial_\nu w + w\kappa)V dS + \int_\Gamma (\partial_\nu H + H\kappa)V dS.$$

By letting $I_{H_p} = \int_\Gamma H_p(x, I_w)dS$ and reorganizing the terms, we obtain the result.

□

Then the explicit form of the shape gradient is

$$G = (H + I_{H_p} w)\kappa + \partial_\nu H + I_{H_p}\partial_\nu w,$$

and has the form (1.13) with $g = \left(H(x, I_w) + I_{H_p} w(x)\right)$ and $f = \partial_\nu H + I_{H_p}\partial_\nu w$.

Theorem 2.2.9. *The second shape derivative of* (1.9) *is given by*

$$\begin{aligned} d^2J(\Gamma;V,W) &= \int_\Gamma (H + I_{H_p} w)\nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \int_\Gamma \left(\partial_{\nu\nu} H + I_{H_p}\partial_{\nu\nu} w + 2(\partial_\nu H + I_{H_p}\partial_\nu w)\kappa\right) VW dS \\ &+ \int_\Gamma (\partial_\nu H_p + H_p\kappa)\, V dS \int_\Gamma (\partial_\nu w + w\kappa)\, W dS \\ &+ \int_\Gamma (\partial_\nu w + w\kappa)\, V dS \int_\Gamma (\partial_\nu H_p + H_p\kappa)\, W dS \\ &+ I_{H_{pp}} \int_\Gamma (\partial_\nu w + w\kappa)\, V dS \int_\Gamma (\partial_\nu w + w\kappa)\, W dS. \end{aligned}$$

Proof. We use Theorem 2.1.3 with $\psi = H(x, I_w(\Gamma))$. Then

$$\begin{aligned} \psi'(\Gamma;V) &= H_p \int_\Gamma (\partial_\nu w + w\kappa) V dS, \\ \partial_\nu \psi'(\Gamma;V) &= \partial_\nu H_p \int_\Gamma (\partial_\nu w + w\kappa) V dS, \end{aligned}$$

and

$$\begin{aligned} \psi''(\Gamma;V,W) &= (H_p)' \int_\Gamma (\partial_\nu w + w\kappa) V dS + H_p \left(\int_\Gamma (\partial_\nu w + w\kappa) V dS \right)' \\ &= H_{pp}(x, I_w) \int_\Gamma (\partial_\nu w + w\kappa) V dS \int_\Gamma (\partial_\nu w + w\kappa) W dS \\ &+ H_p(x, I_w) \int_\Gamma w \nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ H_p(x, I_w) \int_\Gamma \left(\partial_{\nu\nu} w + 2\kappa \partial_\nu w + (\kappa^2 - \sum \kappa_i^2) w \right) VW dS, \end{aligned}$$

using Lemma 2.1.7. Now we substitute these in (2.44) to obtain

$$\begin{aligned} d^2J(\Gamma;V,W) &= \int_\Gamma H_{pp}dS \int_\Gamma (\partial_\nu w + w\kappa)V dS \int_\Gamma (\partial_\nu w + w\kappa)W dS \\ &+ \int_\Gamma H_p dS \int_\Gamma w\nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \int_\Gamma H_p dS \int_\Gamma \left(\partial_{\nu\nu} w + 2\kappa\partial_\nu w + (\kappa^2 - \sum \kappa_i^2)w\right) VW dS \\ &+ \int_\Gamma (\partial_\nu H_p + H_p\kappa)W dS \int_\Gamma (\partial_\nu w + w\kappa)V dS \\ &+ \int_\Gamma (\partial_\nu H_p + H_p\kappa)V dS \int_\Gamma (\partial_\nu w + w\kappa)W dS \\ &+ \int_\Gamma \left(H\nabla_\Gamma V \cdot \nabla_\Gamma W + \left(\partial_{\nu\nu} H + 2\kappa\partial_\nu H + (\kappa^2 - \sum \kappa_i^2)H\right) VW\right) dS. \end{aligned}$$

Reorganizing the various terms yields the result. □

2.2.3 Shape Energies with PDEs

In this section we consider the Chan-Vese approach to the Mumford-Shah model [20] for a given image intensity function $I \in L^2(D)$:

$$J(\Gamma) = \sum_{i=1}^{2} \frac{1}{2} \int_{\Omega_i} \left((u_i - I)^2 + \mu|\nabla u_i|^2\right) dx + \gamma \int_\Gamma dS \tag{2.45}$$

such that

$$\begin{cases} -\mu\Delta u_i + u_i &= I \quad \text{in} \quad \Omega_i \\ \partial_{\nu_i} u_i &= 0 \quad \text{on} \quad \partial\Omega_i, \end{cases} \tag{2.46}$$

where $\Omega_1 = \Omega \subset \mathbb{R}^2$ is the domain inside the curve Γ and $\Omega_2 = D - \Omega$ is the domain outside Γ. Hintermüller and Ring compute the first and second shape derivatives in [36]. We recall them below. We also review the derivation of the first shape derivative for the sake of completeness.

Theorem 2.2.10. *The first shape derivative of the Mumford-Shah energy* (2.45) *is given by*

$$dJ(\Gamma;V)=\int_\Gamma\Big(\frac{1}{2}\left[\!\left[|u-I|^2\right]\!\right]+\frac{\mu}{2}\left[\!\left[|\nabla_\Gamma u|^2\right]\!\right]+\gamma\kappa\Big)VdS.$$

where $[\![f]\!]=f_1-f_2$ *stands for the jump of* f *across* Γ.

Proof. Consider $J(\Omega)=\sum_{i=1,2}J_i(\Omega_i)+J_3(\Gamma)$ with

$$J_i(\Omega_i)=\frac{1}{2}\int_{\Omega_i}\left((u_i-I)^2+\mu|\nabla u_i|^2\right)dx,\qquad J_3(\Gamma)=\gamma\int_\Gamma dS.$$

First shape derivative of J_2 follows from Lemma 2.1.2

$$J_3(\Gamma;V)=\gamma\int_\Gamma\kappa VdS.$$

Since in our convention $\Omega_1=\Omega$ and $\Omega_2=D-\Omega$, we have $V_1=V$, $V_2=-V$. Then to calculate $dJ_i(\Omega_i;V_i)$, we use Theorem 2.1.1 and we let $\phi_i(x,\Omega_i)=\frac{1}{2}(u_i-I)^2+\frac{\nu}{2}|\nabla u_i|^2$ so that

$$\phi'(\Omega_i;V_i)=(u_i-I)u_i'+\mu\nabla u_i\cdot\nabla u_i',$$

where $u_i'=u_i'(\Omega_i;V_i)$. So from Theorem 2.1.1 it follows that

$$dJ_1(\Omega_i;V_i)=\int_{\Omega_i}\left((u_i-I)u_i'+\mu\nabla u_i\cdot\nabla u_i'\right)dx+\frac{1}{2}\int_\Gamma\left((u_i-I)^2+\mu|\nabla u_i|^2\right)VdS.$$

Also note $\nabla u_i=\nabla_\Gamma u_i$ since $\partial_{\nu_i}u_i=0$ on Γ. So collecting all terms we have

$$\begin{aligned}dJ(\Gamma;V) &= \sum_{i=1}^{2}\int_{\Omega_i}\left((u_i-I)u_i'+\mu\nabla u_i\cdot\nabla u_i'\right)dx \\ &+ \frac{1}{2}\int_\Gamma\left(\left[\!\left[(u-I)^2\right]\!\right]+\mu\left[\!\left[|\nabla u|^2\right]\!\right]\right)VdS+\gamma\int_\Gamma\kappa VdS.\end{aligned}\tag{2.47}$$

Now let us investigate $u_i'(\Omega_i;V_i)$. Consider the weak form of (2.46)

$$\int_{\Omega_i}(\mu\nabla u_i\cdot\nabla\phi+u_i\phi)dx=\int_{\Omega_i}I\phi dx\tag{2.48}$$

for all $\phi \in H^1(\Omega_i)$ and take the shape derivative (assuming ϕ is shape-independent)

$$\int_{\Omega_i} (\mu \nabla u_i' \cdot \nabla \phi + u_i' \phi) dx + \int_\Gamma (\mu \nabla u_i \nabla \phi + u_i \phi) V_i dS = \int_\Gamma I \phi V_i dx.$$

One can argue that this equation has a unique solution $u_i' \in H^1(\Omega_i)$ and we use it as a test function in (2.48). In this way we see that the first integral in (2.47) vanishes, which leaves us with the sought result. □

Then the explicit form of the shape gradient is

$$G = \frac{1}{2} [\![|u - I|^2]\!] + \frac{\mu}{2} [\![|\nabla_\Gamma u|^2]\!] + \gamma \kappa,$$

and has the form (1.13) with $g = \gamma$ and $f = \frac{1}{2} [\![|u - I|^2]\!] + \frac{\mu}{2} [\![|\nabla_\Gamma u|^2]\!]$.

Theorem 2.2.11. *The second shape derivative of the Mumford-Shah energy* (2.45) *is given by*

$$\begin{aligned} d^2 J(\Gamma; V, W) &= \gamma \int_\Gamma \nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \frac{\mu}{2} \int_\Gamma \left(\kappa [\![|\nabla u|^2]\!] + \partial_\nu [\![|\nabla u|^2]\!] \right) V W dS \\ &+ \frac{1}{2} \int_\Gamma \left(\kappa [\![|u - I|^2]\!] + \partial_\nu [\![|u - I|^2]\!] \right) V W dS \\ &+ \int_\Gamma \left([\![(u - I) u_W']\!] + \mu [\![\nabla u \cdot \nabla u_W']\!] \right) V dS, \end{aligned}$$

where $u_{i,W}' = u_i'(\Omega_i; W)$, the shape derivative of u_i in Ω_i with respect to the velocity field W, satisfies

$$\begin{cases} -\mu \Delta u_{i,W}' + u_{i,W}' &= 0 \quad \text{in} \quad \Omega_i \\ \frac{\partial u_{i,W}'}{\partial \nu} &= \operatorname{div}_\Gamma (W \nabla_\Gamma u_i) + \frac{1}{\mu} (I - u_i) W \quad \text{on} \quad \partial \Omega_i. \end{cases}$$

Chapter 3

Discrete Gradient Flows

Now we start describing a discrete strategy to address the shape optimization models that we presented in §1. Our goal is to build a sequence of shapes $\{\Omega_n\}_{n\geqslant 0}$ such that $J(\Omega_{n+1}) \leqslant J(\Omega_n)$. First we will elaborate on the choice of gradient descent directions available for energy minimization. This choice along with some geometric relationships will result in system of PDEs. Then we will introduce the appropriate time discretization for this system. At this point we should remark that the shape optimization problem is inherently discrete in time. Unlike the approximation of geometric flows, we do not consider the behavior as the time step goes to zero, because this would serve against our goal of reaching the optimal shape with as few iterations as possible. Then having the time discretization and an appropriate time step selection strategy, we will describe a space discretization using the finite element method. We will also comment on the resulting linear system and the solution strategy.

3.1 Choosing the Descent Direction

Let us recall that for a given shape functional $J(\Omega)$, the shape derivative would be of the form

$$dJ(\Omega; V) = \int_\Gamma GV dS, \tag{3.1}$$

where we prefer to write the shape gradient G as

$$G = g(\Omega)\kappa + f(\Omega). \tag{3.2}$$

In order to do the minimization, the usual choice in the image processing community is the so-called *gradient descent* direction

$$V = -g(\Omega)\kappa - f(\Omega). \tag{3.3}$$

It is easy to show that this choice gives $dJ(\Omega; V) \leqslant 0$. However this descent direction is not the only possible choice. If we take a Hilbert space $B(\Gamma)$ of scalar functions defined on Γ and the associated scalar product $b(\cdot, \cdot)$, we can solve the following equation to obtain other descent directions

$$b(V, W) = -dJ(\Gamma; W), \quad \forall W \in B(\Gamma). \tag{3.4}$$

If $\mathcal{B}$ is an (elliptic) operator such that $\langle \mathcal{B}V, W \rangle = b(V, W)$, this is equivalent to solving the following elliptic PDE on Γ

$$\mathcal{B}V = -g(\Omega)\kappa - f(\Omega). \tag{3.5}$$

Again one can show that the velocity V computed this way satisfies

$$dJ(\Gamma; V) = -b(V, V) = -\|V\|^2_{B(\Gamma)} \leqslant 0, \tag{3.6}$$

where $\| \cdot \|^2_{B(\Gamma)}$ is the norm induced by $b(\cdot, \cdot)$.

A straight-forward choice for $b(\cdot, \cdot)$ is the $L^2(\Gamma)$ scalar product. This gives

$$\langle V, W \rangle = -\int_\Gamma GW dS, \quad \forall W \in B(\Gamma). \tag{3.7}$$

Note the usual choice (3.3) in image processing corresponds to the strong form of the L^2 gradient flow. A possible drawback with this choice is related to the well-posedness of the descent direction (3.3), because (3.3) is a backward-forward parabolic-type equation, depending on the sign of the function $g(\Omega)$. It is locally backward (ill-posed) in regions where $g < 0$ and forward otherwise.

This brings us to a second possibility; we can choose $b(\cdot, \cdot)$ to coincide with a weighted $H^1(\Gamma)$ scalar product. Then we solve

$$\langle \alpha \nabla_\Gamma V, \nabla_\Gamma W \rangle + \langle \beta V, W \rangle = -\int_\Gamma G W dS, \quad \forall W \in B(\Gamma), \tag{3.8}$$

where $\alpha = \alpha(x, \Omega)$, $\beta = \beta(x, \Omega)$ are some positive functions. This equation in the strong form would be

$$-\mathrm{div}_\Gamma(\alpha \nabla_\Gamma V) + \beta V = -g(\Omega)\kappa - f(\Omega). \tag{3.9}$$

One benefit of this scalar product is that we can use this to stabilize an ill-posed L^2 gradient flow with a suitable choice of α and β. Moreover we observe that the bilinear forms given by the second shape derivatives are often in the form of weighted H^1 inner products or can be adjusted to be in this form. These result in inexact Newton-type optimization methods.

A third possibility is to choose $b(\cdot, \cdot)$ to coincide with the $H^{-1}(\Gamma)$ scalar product. If we choose $g(\Omega) = \gamma = \text{const}$ for simplicity, the corresponding velocity equation in the strong form would be

$$V = -\gamma \Delta_\Gamma \kappa + \Delta_\Gamma f. \tag{3.10}$$

The geometric law $V = \Delta_\Gamma \kappa$ is called surface diffusion. This is pursued in detail

in [11] and [28]. By integrating this over Γ, it is easy to see that this flow has the volume preserving property. However the advantage of this choice for the image segmentation models is not obvious. Therefore we will not pursue it. See [15] for the possibility of using other scalar products.

Given a surface Γ, (3.4) by itself is not enough. We need a way to approximate the mean curvature κ and eventually we need a vector velocity $\vec{V}$ to move the surface. Recalling some basic differential geometry (as in [28]), we introduce the following system of differential equations

$$\vec{\kappa} = -\Delta_\Gamma \vec{X} \tag{3.11}$$

$$\kappa = \vec{\kappa} \cdot \vec{\nu} \tag{3.12}$$

$$\mathcal{B}V = -g(\Omega)\kappa - f(\Omega) \tag{3.13}$$

$$\vec{V} = V\vec{\nu}. \tag{3.14}$$

In the equation for $\vec{\kappa}$, we use the minus sign to be consistent with the convention that a circle with outward unit normal ν has positive curvature. Now solving this system of PDEs, we can compute $\vec{V}$ to evolve the surface. The first identity in (3.11) has been first used by G. Dziuk [29] to discretize the mean curvature flow.

3.2 Time Discretization

We will now describe the time discretization for the gradient flows. Let us point out once more that the gradient descent process is inherently discrete in time. Hence our statements about the time discretization refer to a discretization choice rather than an approximation in time. The choice is between an explicit and a semi-

implicit scheme. By explicit and implicit, we mean whether the velocity and the curvature (also possibly the geometric operators $\mathrm{div}_\Gamma, \nabla_\Gamma, \langle\cdot,\cdot\rangle_\Gamma$) will be computed on the current surface Γ_n or the next surface Γ_{n+1}. Each has its advantages and disadvantages and we can choose depending on our application.

3.2.1 The Explicit Case

Given the system of equations (3.11) to compute $\vec{V}$, one straight-forward approach is to assume $\vec{X}$ (the current surface) given and to compute as follows: $\vec{\kappa}_n \leftarrow \vec{X}_n$, $\kappa_n \leftarrow \vec{\kappa}_n$, $V_n \leftarrow (\kappa_n, \vec{X}_n)$, finally $\vec{V}_n \leftarrow V_n$ and to update the surface with $\vec{X}_{n+1} = \vec{X}_n + \tau\vec{V}_n$, where τ is the time step. While this is a relatively efficient way of computing the velocity, it may be prone to stability issues. This is because the velocity equation (3.5) includes the mean curvature term, also known as the *geometric diffusion* term. Note however that using a weighted $H^1(\Gamma)$ scalar product may have stabilizing effect on the flow. So for practical purposes we propose the explicit time discretization

$$\vec{\kappa}_n = -\Delta_{\Gamma_n}\vec{X}_n \tag{3.15}$$

$$\kappa_n = \vec{\kappa}_n \cdot \vec{\nu}_n \tag{3.16}$$

$$\mathcal{B}_n V_n = -g(\Omega_n)\kappa_n - f(\Omega_n) \tag{3.17}$$

$$\vec{V}_n = V_n\vec{\nu}_n. \tag{3.18}$$

Then the update for the surface is

$$\vec{X}_{n+1} = \vec{X}_n + \tau_n\vec{V}_n. \tag{3.19}$$

3.2.2 The Semi-Implicit Case

We will now describe the semi-implicit time discretization. We will show that it has desirable stability properties. We will start with an implicit time discretization following the ideas of Luckhaus [43] and Almgren et al. [3]. For this, we aim to compute the velocity and all geometric quantities on Γ_{n+1}. As this proves to be an impractical endeavour, we then opt for a semi-implicit scheme by introducing an explicit linearization.

An implicit scheme requires us to specify the time step τ_n before we compute the velocity. Still whether τ_n is the best choice or even an energy-decreasing choice is not obvious a priori. But we show that the sequence of pairs $\{(\tau_n, V_{n+1})\}_{n\geq 0}$ computed in this way with the implicit time discretization ensure energy decrease property of the sequence $\{\Omega_n\}_{n\geq 0}$. Still for the semi-implicit case we may need to test several time step candidates to enforce energy decrease.

3.2.2.1 Implicit Time Discretization

We start with the implicit time discretization. Given a variable time step τ_n, let the domain Ω_{n+1} be the solution of the following penalized minimization problem:

$$\text{argmin}_{\Omega_{n+1}}\Big(J(\Omega_{n+1}) + \frac{1}{2\tau_n} d^2(\Omega_{n+1}, \Omega_n)\Big), \tag{3.20}$$

where the "distance term" $\frac{1}{2\tau_n} d^2(\Omega_{n+1}, \Omega_n)$ penalizes the distance between Ω_{n+1} and Ω_n. In order to specify the distance function $d(\cdot, \cdot)$, we consider $\vec{V}_{n+1} := \vec{V}(\vec{X}_{n+1})$

to be the *implicit* Euler approximation of (2.3):

$$\vec{X}_{n+1} = \vec{X}_n + \tau_n \vec{V}_{n+1}; \tag{3.21}$$

Note that Ω_{n+1} is described by the set of points $\vec{X}_{n+1}$ and that $\vec{V}_{n+1}$ is defined in Γ_{n+1}, so (3.21) does not specify $\vec{V}_{n+1}$ directly.

Let $V_{n+1} \in B(\Gamma_{n+1})$. Then a natural choice

$$d(\Omega_{n+1}, \Omega_n) = \|\tau_n V_{n+1}\|_{B(\Gamma_{n+1})},$$

converts (3.20) into the following minimization problem for $\vec{V}_{n+1} = V_{n+1}\vec{\nu}_{n+1}$:

$$\text{argmin}_{\vec{V}_{n+1}} \Big(J(\Omega_n + \tau_n \vec{V}_{n+1}) + \frac{1}{2\tau_n} \|\tau_n V_{n+1}\|^2_{B(\Gamma_{n+1})} \Big). \tag{3.22}$$

The optimality condition reads as follows

$$dJ(\Omega_{n+1}; \tau_n \vec{W}) + \frac{1}{\tau_n} b_{n+1}(\tau_n V_{n+1}, \tau_n W) = 0, \quad \forall W \in B(\Gamma_{n+1}), \tag{3.23}$$

in terms of the variation $\vec{W} = W\vec{\nu}_{n+1}$ of $\vec{V}_{n+1}$. Such a condition, via Hadamard-Zolésio Theorem 2.1.2, is equivalent to

$$\langle G_{n+1}, \tau_n W \rangle_{\Gamma_{n+1}} = -\frac{1}{\tau_n} b_{n+1}(\tau_n V_{n+1}, \tau_n W), \quad \forall W \in B(\Gamma_{n+1}).$$

This yields the following ideal equation for V_{n+1}

$$b_{n+1}(V_{n+1}, W) = -\langle G_{n+1}, W \rangle_{\Gamma_{n+1}}, \quad \forall W \in B(\Gamma_{n+1}), \tag{3.24}$$

which is *implicit* in that the domain Ω_{n+1} is unknown and thus part of the problem. A crucial consequence of (3.22) important for theory is

$$J(\Omega_{n+1}) = J(\Omega_n + \tau_n \vec{V}_{n+1}) \leq J(\Omega_n + \tau_n \vec{V}_{n+1}) + \frac{\tau_n}{2} \|V_{n+1}\|^2_{B(\Gamma_{n+1})} \leq J(\Omega_n), \tag{3.25}$$

as results from taking $\vec{V}_{n+1} = 0$ in (3.22). Consequently,

$$J(\Omega_k) + \frac{1}{2}\sum_{i=1}^{k} \tau_{i-1} \|\vec{V}_i\|^2_{B(\Gamma_i)} \leq J(\Omega_0), \qquad \forall k \geq 1.$$

Solving the nonlinear problem (3.24) is unaffordable directly and would require iteration. The following linearization technique may either replace the implicit solve or be used as one step in an iterative process.

3.2.2.2 Explicit Linearization

The key idea is to replace in (3.24) the new surface Γ_{n+1}, which is unknown, by the current one Γ_n. This gives rise to the following elliptic PDE on Γ_n: find $V_{n+1} \in B(\Gamma_n)$ such that

$$b_n(V_{n+1}, W) = -\langle G_n, W \rangle_{\Gamma_n}, \qquad \forall W \in B(\Gamma_n). \tag{3.26}$$

Then Γ_{n+1} results from the explicit update $\vec{X}_{n+1} = \vec{X}_n + \tau \vec{V}_{n+1}$, but the energy decrease property (3.25) is no longer valid. Nevertheless, (3.26) still provides a weaker energy decrease property that follows from (3.6). Hence it is possible to obtain energy decrease by choosing the time step small enough.

3.2.2.3 Semi-Implicit Time Discretization

To derive an effective algorithm, we still need to address one critical issue:

Computing geometric quantities such as curvature and normal velocity implicitly, but most of the geometry explicitly, thereby reaching a compromise between the schemes of §§3.2.2.1 and 3.2.2.2; (3.27)

To deal with (3.27) we let $\mathcal{B}_n$ denote the linear operator defined by the scalar product $b_n(\cdot,\cdot)$ on Γ_n, namely,

$$\langle \mathcal{B}_n V, W\rangle = b_n(V,W) \qquad \forall V, W \in B(\Gamma_n).$$

Recalling the special form (3.2) of G, we thus propose the following *semi-implicit* computation of V_{n+1} and κ_{n+1}:

$$\mathcal{B}_n V_{n+1} + g(\Omega_n)\kappa_{n+1} = -f(\Omega_n). \tag{3.28}$$

For consistency with (3.28) we enforce the geometric relationships semi-implicitly

$$-\Delta_{\Gamma_n}\vec{X}_{n+1} = \vec{\kappa}_{n+1}, \qquad \kappa_{n+1} = \vec{\kappa}_{n+1}\cdot\vec{\nu}_n, \qquad \vec{V}_{n+1} = V_{n+1}\vec{\nu}_n.$$

To close the system we must relate position $\vec{X}_{n+1}$ of Γ_{n+1} and velocity $\vec{V}_{n+1}$. We impose

$$\vec{X}_{n+1} = \vec{X}_n + \tau_n \vec{V}_{n+1}, \tag{3.29}$$

whence we get the *semi-implicit scheme*: find $(\vec{\kappa}_{n+1}, \kappa_{n+1}, V_{n+1}, \vec{V}_{n+1})$ such that

$$\vec{\kappa}_{n+1} + \tau_n \Delta_{\Gamma_n}\vec{V}_{n+1} = -\Delta_{\Gamma_n}\vec{X}_n \tag{3.30}$$

$$\kappa_{n+1} - \vec{\kappa}_{n+1}\cdot\vec{\nu}_n = 0 \tag{3.31}$$

$$\mathcal{B}_n V_{n+1} + g(\Omega_n)\kappa_{n+1} = -f(\Omega_n) \tag{3.32}$$

$$\vec{V}_{n+1} - V_{n+1}\vec{\nu}_n = 0. \tag{3.33}$$

3.2.3 Choosing the Time Step

We have said that we update the surface as follows once we have computed the velocity $\vec{V}$:

$$\vec{X}_{n+1} = \vec{X}_n + \tau\vec{V}. \tag{3.34}$$

However we have not explained how we choose the value of τ to ensure $J(\Omega_{n+1}) \leqslant J(\Omega_n)$ or better. On the other hand, since we have $dJ(\Omega_n; \vec{V}) \leqslant 0$, then

$$J(\Omega_{n+1}) \leqslant J(\Omega_n) + \tau dJ(\Omega_n; \vec{V}) + o(\tau) \leqslant J(\Omega_n) \tag{3.35}$$

for small enough τ. Hence one time step selection strategy would be to use very small time steps. Obviously this would result in too many computations. In any case, how small the time step should be is not always easy to figure out. In particular, a time step that seems very small at the beginning of the optimization may turn out to be too big close to the optimum. For these reasons, we describe two classic time step selection schemes from nonlinear optimization. They both satisfy one crucial property: they guarantee energy decrease at each step and result in global convergence.

3.2.3.1 Backtracking

The idea of backtracking is very simple. We take the current time step and check how it changes the energy. If it increases the energy, we bisect the time step and check again. We do this until we find a time step that decreases the energy. Note that this strategy results in a sequence of pairs $\{(\tau_n, \Omega_n)\}$ such that not only $J(\Omega_n)$, but also τ_n always decreases. It may be desirable to allow the time step to increase too. In this case we start checking with twice the current time step. See Algorithm 1.

Algorithm 1 : Backtracking

compute $J_n = J(\Gamma_n)$
let $\tau = \tau_n$ or $2\tau_n$
$iter = 0$
repeat
  $iter = iter + 1$
  $\tilde{\Gamma}_{n+1} = \Gamma_n + \tau\vec{V}$
  compute $\tilde{J}_{n+1} = J(\tilde{\Omega}_{n+1})$
  if $\tilde{J}_{n+1} \geqslant J_n$ **then**
    $\tau = \tau/2$
    recompute $\vec{V}$ if $\vec{V}$ depends on τ
  end if
until $\tilde{J}_{n+1} < J_n$ or $iter > maxiter$

3.2.3.2 Line Search

Another classic approach to ensure energy decrease at each iteration is to perform a *line search* over possible time steps. The method we describe is based on the Armijo-Goldstein rule (see [50]). The method, in essence, tries to balance the expected decrease in energy, Δ_{exp}, with the effective decrease in energy, Δ_{eff}, where

$$\Delta_{exp} = \tau dJ(\Omega_n; V), \qquad \Delta_{eff} = J(\Omega_{n+1}) - J(\Omega_n). \tag{3.36}$$

For this, at each iteration we try to choose a time step, τ, such that

$$\alpha\Delta_{exp} \leqslant \Delta_{eff} \leqslant \beta\Delta_{exp} \tag{3.37}$$

is satisfied, with $0 < \alpha < \beta < 1$. We give the details of the method in Algorithm 2.

3.3 Finite Element Discretization

We now discuss the finite element discretization of (3.15)-(3.18) and (3.30)-(3.33). Let $\mathcal{T} = \mathcal{T}_n$ be a shape-regular but possibly graded mesh of triangular finite

Algorithm 2 : Line search

choose $0 < \alpha < \beta < 1$ and $0 < \epsilon \ll \frac{1}{2}$
$iter = 0$
$a = \tau_{min}$, $b = \tau_{max}$
repeat
  $r = b - a$
  choose $\tilde{\tau} \in (a + \epsilon r, b - \epsilon r)$
  $\tilde{\Gamma}_{n+1} = \Gamma_n + \tau \vec{V}$
  compute $\tilde{J}_{n+1} = J(\tilde{\Omega}_{n+1})$
  let $\Delta_{eff} = \tilde{J}_{n+1} - J(\Omega_n)$
  let $\Delta_{exp} = \tilde{\tau} dJ(\Omega_n; V)$
  if $\Delta_{eff} > \Delta_{exp}$ **then**
    $b = \tilde{\tau}$
  else if $\Delta_{eff} < \Delta_{exp}$ **then**
    $a = \tilde{\tau}$
  else
    $\tau = \tilde{\tau}$
  end if
  $iter = iter + 1$
until $\alpha\Delta_{exp} \leqslant \Delta_{eff} \leqslant \beta\Delta_{exp}$ or $iter > maxiter$

elements over the surface $\Gamma = \Gamma_n$, which, from now on, is assumed to be polyhedral. To simplify the notations we hereafter drop the subscripts n and $n+1$. Let $T \in \mathcal{T}$ be a typical triangle and let $\vec{\nu}_T = (\nu_T^i)_{i=1}^d$ be the unit normal to T pointing outwards. We denote by $\vec{\nu}$ the outward unit normal to Γ, which satisfies $\vec{\nu}|_T = \vec{\nu}_T$ for all $T \in \mathcal{T}$, and is thus discontinuous across inter-element boundaries. Let $\{\phi_i\}_{i=1}^I$ be the set of canonical basis functions of the finite element space $\mathcal{V}(\Gamma)$ of continuous piecewise polynomials P^k of degree $\leq k$ over $\mathcal{T}$ for $k \geq 1$; we also set $\vec{\mathcal{V}}(\Gamma) := \mathcal{V}(\Gamma)^d$. We thus have a conforming approximation $\mathcal{V}(\Gamma)$ of $H^1(\Gamma)$.

3.3.1 The Explicit Case

We multiply equations (3.15)-(3.18) by test functions $\phi \in \mathcal{V}(\Gamma)$ and $\vec{\phi} \in \vec{\mathcal{V}}(\Gamma)$ and integrate by parts those terms involving Δ_Γ. We thus arrive at the fully discrete problem: seek $\vec{V}, \vec{\kappa} \in \vec{\mathcal{V}}(\Gamma)$, $V, \kappa \in \mathcal{V}(\Gamma)$, such that

$$\langle \vec{\kappa}, \vec{\phi} \rangle = \langle \nabla_\Gamma \vec{X}, \nabla_\Gamma \vec{\phi} \rangle \qquad \forall \vec{\phi} \in \vec{\mathcal{V}}(\Gamma), \tag{3.38}$$

$$\langle \kappa, \phi \rangle = \langle \vec{\kappa} \cdot \vec{\nu}, \phi \rangle \qquad \forall \phi \in \mathcal{V}(\Gamma), \tag{3.39}$$

$$\langle \alpha \nabla_\Gamma V, \nabla_\Gamma \phi \rangle + \langle \beta V, \phi \rangle = -\langle g\kappa, \phi \rangle - \langle f, \phi \rangle \quad \forall \phi \in \mathcal{V}(\Gamma), \tag{3.40}$$

$$\langle \vec{V}, \vec{\phi} \rangle = \langle V, \vec{\phi} \cdot \vec{\nu} \rangle \qquad \forall \vec{\phi} \in \vec{\mathcal{V}}(\Gamma). \tag{3.41}$$

3.3.2 The Semi-Implicit Case

We multiply equations (3.30)-(3.33) by test functions $\phi \in \mathcal{V}(\Gamma)$ and $\vec{\phi} \in \vec{\mathcal{V}}(\Gamma)$ and again integrate by parts. We obtain the fully discrete problem: seek $\vec{V}, \vec{\kappa} \in \vec{\mathcal{V}}(\Gamma)$, $V, \kappa \in \mathcal{V}(\Gamma)$, such that

$$\langle \vec{\kappa}, \vec{\phi} \rangle - \tau \langle \nabla_\Gamma \vec{V}, \nabla_\Gamma \vec{\phi} \rangle = \langle \nabla_\Gamma \vec{X}, \nabla_\Gamma \vec{\phi} \rangle \quad \forall \vec{\phi} \in \vec{\mathcal{V}}(\Gamma), \tag{3.42}$$

$$\langle \kappa, \phi \rangle - \langle \vec{\kappa} \cdot \vec{\nu}, \phi \rangle = 0 \qquad \forall \phi \in \mathcal{V}(\Gamma), \tag{3.43}$$

$$\langle \alpha \nabla_\Gamma V, \nabla_\Gamma \phi \rangle + \langle \beta V, \phi \rangle + \langle g\kappa, \phi \rangle = -\langle f, \phi \rangle \qquad \forall \phi \in \mathcal{V}(\Gamma), \tag{3.44}$$

$$\langle \vec{V}, \vec{\phi} \rangle - \langle V, \vec{\phi} \cdot \vec{\nu} \rangle = 0 \qquad \forall \vec{\phi} \in \vec{\mathcal{V}}(\Gamma). \tag{3.45}$$

3.4 Solvability and Stability

We will now show the well-posedness of the linear systems resulting from the finite element discretization. We will also provide some stability results. Let us first

give the following lemma, which will be useful to prove the results.

Lemma 3.4.1 (Area inequality [10]). *Let $d = 2, 3$ and Γ, $\tilde{\Gamma}$ be $d-1$–dimensional, closed, regular $C^{0,1}$–manifolds embedded in $\mathbb{R}^d$. Moreover let $\vec{Y} : \Gamma \to \tilde{\Gamma}$ be a homeomorphism with $D\vec{Y}$, $(D\vec{Y})^{-1} \in L^\infty$. Then, if $\vec{X}$ denotes the position vector of the integration variable, the following inequality holds:*

$$\langle \nabla_\Gamma \vec{Y}, \nabla_\Gamma (\vec{Y} - \vec{X}) \rangle \;\geqslant\; |\vec{Y}(\Gamma)| \;-\; |\Gamma|.$$

The proof of the above lemma is rather technical and can be found in [10].

3.4.1 The Explicit Case

Theorem 3.4.1. *The system of equations* (3.38)-(3.41) *corresponding to the finite element discretization of the explicit case has a unique solution.*

Proof. We consider (3.38)-(3.41) with the right hand side equal to zero

$$\langle \vec{\kappa}, \vec{\phi} \rangle = 0 \quad \forall \vec{\phi} \in \vec{\mathcal{V}}(\Gamma), \tag{3.46}$$

$$\langle \kappa, \phi \rangle = 0 \quad \forall \phi \in \mathcal{V}(\Gamma), \tag{3.47}$$

$$\langle \alpha \nabla_\Gamma V, \nabla_\Gamma \phi \rangle + \langle \beta V, \phi \rangle = 0 \quad \forall \phi \in \mathcal{V}(\Gamma), \tag{3.48}$$

$$\langle \vec{V}, \vec{\phi} \rangle = 0 \quad \forall \vec{\phi} \in \vec{\mathcal{V}}(\Gamma). \tag{3.49}$$

It suffices to show that the solution is 0. Testing (3.46) with $\vec{\phi} = \vec{\kappa}$ gives $\|\vec{\kappa}\| = 0$.

Testing (3.47) with $\phi = \kappa$ gives $\|\kappa\| = 0$.

Testing (3.48) with $\phi = V$ gives $0 \leqslant \min_\Gamma(\alpha, \beta)\|V\|_{H^1(\Gamma)} \leqslant \langle \alpha \nabla_\Gamma V, \nabla_\Gamma V \rangle + \langle \beta V, V \rangle = 0$.

Testing (3.49) with $\vec{\phi} = \vec{V}$ gives $\|\vec{V}\| = 0$.

□

3.4.2 The Semi-Implicit Case

We prove that the semi-implicit scheme is unconditionally stable for the L^2 flow, provided that $g \geqslant m_g > 0$.

Theorem 3.4.2 (Unconditional Stability for L^2 flow). *Let $\alpha = 0$, $\beta = 1$, $g \geqslant m_g > 0$. Let $(V^n, \kappa^n, \vec{V}^n, \vec{\kappa}^n)_{n=1}^N$ be the solution of the fully discrete equations (3.42)–(3.45) and let Γ^n be the corresponding embedded surfaces. Then for all $1 \leq m \leq N$ we have*

$$|\Gamma^m| + \frac{1}{2} m_g \sum_{n=0}^{m-1} \tau_n \|\kappa^{n+1}\|^2_{L^2(\Gamma^n)} \leq |\Gamma^0| + \frac{1}{2m_g} \sum_{n=0}^{m-1} \tau_n \|f^n\|^2_{L^2(\Gamma^n)} \tag{3.50}$$

Proof. We start by testing (3.44) with $\phi = \kappa^{n+1}$, thereby obtaining

$$\langle V^{n+1}, \kappa^{n+1} \rangle + \langle g^n \kappa^{n+1}, \kappa^{n+1} \rangle = -\langle f^n, \kappa^{n+1} \rangle$$

Combining (3.45) with $\vec{\phi} = \vec{\kappa}^{n+1}$ and (3.43) with $\phi = V^{n+1}$, we easily arrive at

$$\langle \vec{V}^{n+1}, \vec{\kappa}^{n+1} \rangle = \langle V^{n+1}, \vec{\kappa}^{n+1} \cdot \vec{\nu} \rangle = \langle \kappa^{n+1}, V^{n+1} \rangle,$$

whence

$$\langle \vec{V}^{n+1}, \vec{\kappa}^{n+1} \rangle + \langle g^n \kappa^{n+1}, \kappa^{n+1} \rangle = -\langle f^n, \kappa^{n+1} \rangle. \tag{3.51}$$

On the other hand, testing (3.42) with $\vec{\phi} = \tau_n \vec{V}^{\,n+1}$ and observing that, according to (3.29), $\tau_n \vec{V}^{n+1} = \vec{X}^{n+1} - \vec{X}^n$ yields

$$\tau_n \langle \vec{V}^{n+1}, \vec{\kappa}^{n+1} \rangle - \langle \nabla_\Gamma \vec{X}^{n+1}, \nabla_\Gamma (\vec{X}^{n+1} - \vec{X}^n) \rangle = 0. \tag{3.52}$$

Multiplying (3.51) by τ_n and substituting into (3.52) we infer that

$$\langle \nabla_\Gamma \vec{X}^{n+1}, \nabla_\Gamma(\vec{X}^{n+1} - \vec{X}^n)\rangle + \tau_n \langle g^n \kappa^{n+1}, \kappa^{n+1}\rangle = -\langle f^n, \kappa^{n+1}\rangle.$$

Since we have $g \geqslant m_g > 0$, we can write

$$\langle \nabla_\Gamma \vec{X}^{n+1}, \nabla_\Gamma(\vec{X}^{n+1} - \vec{X}^n)\rangle + \tau_n m_g \|\kappa^{n+1}\|^2 \leqslant -\tau_n \langle f^n, \kappa^{n+1}\rangle.$$

Applying the inequality $ab \leqslant \epsilon a^2 + \frac{1}{4\epsilon} b^2$ with $\epsilon = \frac{m_g}{2}, \quad a = \kappa^{n+1}, \quad b = -f^n$

$$\langle \nabla_\Gamma \vec{X}^{n+1}, \nabla_\Gamma(\vec{X}^{n+1} - \vec{X}^n)\rangle + \tau_n m_g \|\kappa^{n+1}\|^2 \leqslant \tau_n \frac{m_g}{2} \|\kappa^{n+1}\|^2 + \frac{\tau_n}{2m_g} \|f^n\|^2$$

whence, using Lemma 3.4.1,

$$|\Gamma^{n+1}| - |\Gamma^n| + \tau_n \frac{m_g}{2} \|\kappa^{n+1}\|^2 \leqslant \frac{\tau_n}{2m_g} \|f^n\|^2.$$

Summing up over n, from 0 to $m-1$, yields the asserted result. □

3.5 Matrix Formulation

We turn our attention to an equivalent matrix formulation to the fully discrete problem. Given the matrix entries

$$M_{g_{i,j}} := \langle g\phi_i, \phi_j\rangle, \quad M_{\beta_{i,j}} := \langle \beta\phi_i, \phi_j\rangle, \quad M_{i,j} := \langle \phi_i, \phi_j\rangle, \quad \vec{M}_{i,j} := M_{i,j}\vec{Id},$$

$$\vec{N}_{i,j} := (N^k_{i,j})_{k=1}^d := \langle \phi_i, \phi_j \nu^k\rangle_{k=1}^d,$$

$$A_{i,j} := \langle \nabla_\Gamma \phi_i, \nabla_\Gamma \phi_j\rangle, \quad A_{\alpha i,j} := \langle \alpha \nabla_\Gamma \phi_i, \nabla_\Gamma \phi_j\rangle, \quad \vec{A}_{i,j} := A_{i,j}\vec{Id},$$

with $\vec{Id} \in \mathbb{R}^{d\times d}$ being the identity matrix and $(\vec{e}_k)_{k=1}^d$ the canonical basis of $\mathbb{R}^d$, the mass and stiffness matrices are

$$M_g := (M_{g_{i,j}})_{i,j=1}^I, \quad M_\beta := (M_{\beta_{i,j}})_{i,j=1}^I, \quad M := (M_{i,j})_{i,j=1}^I, \quad \vec{M} := (\vec{M}_{i,j})_{i,j=1}^I,$$

$$\vec{N} := (\vec{N}_{i,j})_{i,j=1}^I,$$

$$A_\alpha := (A_{\alpha i,j})_{i,j=1}^I, \quad A := (A_{i,j})_{i,j=1}^I, \quad \vec{A} := (\vec{A}_{i,j})_{i,j=1}^I.$$

We point out that $\vec{M}, \vec{A}$ and $\vec{N}$ possess matrix-valued entries and therefore the matrix-vector product is understood in the following sense

$$\vec{M}\vec{\mathbf{V}} = \big(\sum_{j=1}^{I} \vec{M}_{i,j}\vec{V}_j\big)_{i=1}^{I},$$

each component $\vec{V}_i$ of $\vec{\mathbf{V}}$, as well as each of $\vec{M}\vec{\mathbf{V}}$, is itself a vector in $\mathbb{R}^d$.

We use the convention that a vector of nodal values of a finite element function is written in bold face: $\mathbf{V} = (V_i)_{i=1}^{I} \in \mathbb{R}^I$ is equivalent to $V = \sum_{i=1}^{I} V_i\phi_i \in \mathcal{V}(\Gamma)$.

3.5.1 The Explicit Case

We are now in a position to write the matrix formulation of (3.38)-(3.41). Upon expanding the unknown scalar functions $V, K \in \mathcal{V}(\Gamma)$ and vector functions $\vec{V}, \vec{K} \in \vec{\mathcal{V}}$ in terms of the basis functions and setting $\phi = \phi_i$ and $\vec{\phi}^k = \phi\vec{e}_k$, we easily arrive at the *linear* system of equations

$$\begin{aligned} \vec{M}\vec{\mathbf{K}} &= \vec{A}\vec{\mathbf{X}} \\ M\mathbf{K} &= \vec{N}^T\vec{\mathbf{K}} \\ (A_\alpha + M_\beta)\mathbf{V} &= -M_g\mathbf{K} - \mathbf{f} \\ \vec{M}\vec{\mathbf{V}} &= \vec{N}\mathbf{V}. \end{aligned} \tag{3.53}$$

3.5.2 The Semi-Implicit Case

Now we write the matrix formulation of (3.42)-(3.45). Upon expanding the unknown scalar functions $V, K \in \mathcal{V}(\Gamma)$ and vector functions $\vec{V}, \vec{K} \in \vec{\mathcal{V}}$ in terms of the basis functions and setting $\phi = \phi_i$ and $\vec{\phi}^k = \phi\vec{e}_k$, we have the following *linear*

system of equations

$$\begin{aligned} -\tau\vec{A}\vec{\mathbf{V}} + \vec{M}\vec{\mathbf{K}} &= \vec{A}\vec{\mathbf{X}} \\ M\mathbf{K} - \vec{N}^T\vec{\mathbf{K}} &= \mathbf{0} \\ (A_\alpha + M_\beta)\mathbf{V} + M_g\mathbf{K} &= -\mathbf{f} \\ \vec{M}\vec{\mathbf{V}} - \vec{N}\mathbf{V} &= \vec{\mathbf{0}}. \end{aligned} \tag{3.54}$$

3.6 Solving the Linear System

3.6.1 The Explicit Case

Given the explicit system (3.53), we can solve for the velocity $\vec{V}$ as follows

$$\begin{aligned} \vec{\mathbf{K}} &= \vec{M}^{-1}\vec{A}\vec{\mathbf{X}} \\ \mathbf{K} &= M^{-1}\vec{N}^T\vec{\mathbf{K}} \\ \mathbf{V} &= (A_\alpha + M_\beta)^{-1}(-M_g\mathbf{K} - \mathbf{f}) \\ \vec{\mathbf{V}} &= \vec{M}^{-1}\vec{N}\mathbf{V}. \end{aligned} \tag{3.55}$$

If $(A_\alpha + M_\beta)$ is well-conditioned, this system of equations can be solved in a straightforward manner. In particular, for curves in 2D, the matrices can be organized in tridiagonal form. Then the time complexity for the whole solution will be linear in the number of elements.

3.6.2 The Semi-Implicit Case

The solution of the semi-implicit system (3.54) is a bit more involved compared to the explicit system. Let us rewrite the system in the following way

$$\begin{pmatrix} \vec{M} & 0 & 0 & -\vec{N} \\ 0 & M & -\vec{N}^T & 0 \\ -\tau\vec{A} & 0 & \vec{M} & 0 \\ 0 & M_g & 0 & A_\alpha + M_\beta \end{pmatrix} \begin{pmatrix} \vec{\mathbf{V}} \\ \mathbf{K} \\ \vec{\mathbf{K}} \\ \mathbf{V} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vec{A}\vec{\mathbf{X}} \\ -\mathbf{f} \end{pmatrix}, \tag{3.56}$$

or equivalently, with obvious notation and $\mathbf{X}_1 = (\vec{\mathbf{V}}, \mathbf{K})^T, \mathbf{X}_2 = (\vec{\mathbf{K}}, \mathbf{V})^T$,

$$\begin{pmatrix} Z & N \\ C & \tilde{A} \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{h} \end{pmatrix}. \tag{3.57}$$

Invoking the equalities

$$\mathbf{X}_1 = -Z^{-1}N\mathbf{X}_2 \tag{3.58}$$

$$(-CZ^{-1}N + \tilde{A})\mathbf{X}_2 = \mathbf{h}, \tag{3.59}$$

we see that (3.59) is equivalent to

$$\begin{pmatrix} \vec{M} & -\tau\vec{A}\vec{M}^{-1}\vec{N} \\ M_g M^{-1}\vec{N}^T & A_\alpha + M_\beta \end{pmatrix} \begin{pmatrix} \vec{\mathbf{K}} \\ \mathbf{V} \end{pmatrix} = \begin{pmatrix} \vec{A}\vec{\mathbf{X}} \\ -\mathbf{f} \end{pmatrix}. \tag{3.60}$$

Finally we write the Schur complement as follows

$$(\tau M_g M^{-1}\vec{N}^T\vec{M}^{-1}\vec{A}\vec{M}^{-1}\vec{N} + A_\alpha + M_\beta)\mathbf{V} = -\mathbf{f} - M_g M^{-1}\vec{N}^T\vec{M}^{-1}\vec{A}\vec{\mathbf{X}}. \tag{3.61}$$

Hence we obtain the velocity by solving this system at each time step. Note that if we multiply (3.61) with MM_g^{-1}, we obtain

$$\left(\tau\vec{N}^T\vec{M}^{-1}\vec{A}\vec{M}^{-1}\vec{N} + MM_g^{-1}(A_\alpha + M_\beta)\right)\mathbf{V} = -MM_g^{-1}\mathbf{f} - \vec{N}^T\vec{M}^{-1}\vec{A}\vec{\mathbf{X}}. \tag{3.62}$$

For the L^2 inner product, we have $\alpha = 0$ and $\beta = 1$, thus

$$\left(\tau \vec{N}^T \vec{M}^{-1} \vec{A} \vec{M}^{-1} \vec{N} + M M_g^{-1} M\right) \mathbf{V} = -M M_g^{-1} \mathbf{f} - \vec{N}^T \vec{M}^{-1} \vec{A} \vec{\mathbf{X}}, \tag{3.63}$$

so that the coefficient matrix of the linear system is symmetric. This is also the case if we have $g = 1$, then

$$\left(\tau \vec{N}^T \vec{M}^{-1} \vec{A} \vec{M}^{-1} \vec{N} + A_\alpha + M_\beta\right) \mathbf{V} = -\mathbf{f} - \vec{N}^T \vec{M}^{-1} \vec{A} \vec{\mathbf{X}}. \tag{3.64}$$

In these cases, since the linear system is symmetric, we can solve them efficiently using the Conjugate Gradient method. On the other hand (3.61) is a nonsymmetric system and we would use GMRES to solve it.

Chapter 4

Computational Enhancements

When we evolve a surface using a Lagrangian approach, maintaining an accurate representation of the geometry becomes an important issue. We may start with a surface of good quality. But the evolution computed, say, for shape optimization may be rough and may deteriorate the mesh representing the surface. We may lose resolution in some regions and the surface elements may cluster in other regions. Even more dramatic effects are possible: nodes of the mesh may become entangled with each other or topological changes may take place, such as merging and splitting of components.

In this chapter, we will describe procedures that will help us address these issues. We will start with some simple procedures and safeguards to maintain the quality of the mesh through the evolution. Then we will describe space adaptivity procedures that automatically adjust the resolution of the mesh to capture the geometry and the data. This will balance accuracy and computational cost. Finally we will give a detailed description of a method to perform topological changes for curves in 2d.

4.1 Maintaining Mesh Quality

In this section we describe three procedures that help us maintain a good quality mesh through the evolution. These are:

1. Time step control: checks the time step against node entanglements,

2. Mesh smoothing: equidistributes the nodes of the surface,

3. Angle width control: a correction method for triangles becoming too thin.

4.1.1 Time Step Control

In general one should avoid taking too large time steps for surface evolution. This is true even though the underlying evolution may be unconditionally stable. Large time steps bring the risks of mesh distortion and mesh entanglement by neighboring nodes crossing each other. To avoid these risks, we present the time step control scheme proposed by Bänsch, Morin and Nochetto in [11], also used in [28]. The main idea of this scheme is not to allow the nodes of an element a relative tangential displacement greater than the size of the element. For this, we take the velocities $\vec{V}_i, \vec{V}_j$ at the i^{th} and j^{th} nodes of the element respectively and compute the relative tangential displacement as

$$\tau \left| (\vec{V}_i - \vec{V}_j) \cdot \vec{t} \right|$$

where τ is the time step and $\vec{t}$ is the unit tangent vector from node i to node j. This quantity should be less than a fraction of h, the local mesh size. Noting that the

tangential gradient $\nabla_\Gamma \vec{V}$ can be approximated by $\frac{(\vec{V}_i - \vec{V}_j)}{h} \cdot \vec{t}$, we obtain the following check for τ

$$\tau |\nabla_\Gamma \vec{V}| \leqslant \epsilon_\tau.$$

Hence ρ, the maximum time step allowed by time step control can be computed by

$$\rho = \frac{\epsilon_\tau}{\max |\nabla_\Gamma \vec{V}|}.$$

The time step control procedure is given in Algorithm 3.

Algorithm 3 : Time step control

compute $\rho = \frac{\epsilon_\tau}{\max |\nabla_\Gamma \vec{V}|}$
if current $\tau > \rho$ **then**
  recompute τ, possibly choosing a fraction of ρ
end if
set $\tau = \max(\tau, \tau_{min})$
set $\tau = \min(\tau, \tau_{max})$

4.1.2 Mesh Smoothing

Ideally, for our Lagrangian method, we would like to have some kind of space adaptivity to guide us in adjusting the resolution of the mesh. Still at initial stages of shape optimization, when we are far away from the optimum, we may opt for uniform resolution, which may be more practical from a computational point of view. The evolution may nevertheless tend to cluster the nodes and deteriorate mesh resolution in certain regions. So we may need some kind of node redistribution algorithm. For this reason, we describe the mesh smoothing procedure introduced by Bänsch, Morin and Nochetto in [11], and further used in [28]. The goal of the method is to equidistribute the nodes of the surface. It has two critical properties:

1) It is realized in the form of a tangential motion and it does not affect normal motion, 2) It is volume-preserving. We describe the mesh smoothing procedure in Algorithm 4. It is executed in the form of Gauss-Seidel updates.

Let us express caution on one aspect of the method. In practice the mesh smoothing procedure tends to perturb the surface slightly. So it is better to switch it off when the surface is close to the optimum. Otherwise it may work against us by continuing to perturb the surface at the optimum. In our experiments we used mesh smoothing only in conjunction with angle width control, which we describe next.

Algorithm 4 : Mesh smoothing

for each node z of the mesh **do**
  compute a normal ν_z to the node z
  compute a weighted average $\hat{z}$ of all vertices of the star centered at z
  take a line l passing through $\hat{z}$ in dir ν_z
  replace z with the point on l keeping the volume unchanged
end for

4.1.3 Angle Width Control

In 3d, the mesh is represented as a triangulated surface and the quality of the triangles becomes an important factor in the accuracy of the computation. Very thin triangles with large obtuse angles create difficulties. For this reason we make use of the angle width control procedure as Bänsch, Morin and Nochetto do in [11], and also Doğan et al. in [28]. This consists of a single splitting of elements with angles wider than a given threshold θ_{max} (120^o in our computations). The newest-vertex bisection rule is used for the splitting. The splittings are followed by n_{MS}

mesh smoothing sweeps. The pseudo-code is given in Algorithm 5.

Algorithm 5 : Angle width control

```
repeat
  mark elements with angles > θ_max
  split all marked elements
  if elements are split then
    perform n_MS mesh smoothing sweeps
  end if
until no more elements are marked
```

4.2 Space Adaptivity

In this section we describe space adaptivity procedures that will help us tune the resolution of the mesh with respect to the geometry and the data. The goal is to balance accuracy and computational cost by refining where resolution is needed and keeping a coarse mesh where fine resolution is not needed. For this, we give three procedures, Algorithms 6, 7 and 8.

Sometimes we need to use more than one of these in conjunction. This happens, for example, in the case of the minimal surface model, where we use both Algorithms 6 and 7. In such cases, the rule is to refine the mesh if any one of the three algorithms signals *refine* and to coarsen if *all* of them signal *coarsen*.

4.2.1 Geometry-driven Adaptivity

The main idea of space adaptivity introduced by Bänsch, Morin and Nochetto in [11] and further used in [28] is to have more nodes where the surface varies more geometrically and to have fewer nodes where the surface varies less. For a surface,

a local measure of variation is the magnitude of the second fundamental form given by $|\nabla_\Gamma \nu|$. We can estimate this quantity as follows

$$|\nabla_\Gamma \nu| = \frac{|\nu_1 - \nu_2|}{h_S} \approx \frac{\theta_S}{h_S},$$

where ν_1, ν_2 are the normals of two adjacent elements $T_1, T_2 \in \mathcal{T}$ sharing a side (node in 2d) S, and θ_S is the angle between ν_1 and ν_2. Recall that the pointwise accuracy of the mesh is proportional to $h_S^2 |\nabla_\Gamma \nu|$. Requiring $h_S^2 |\nabla_\Gamma \nu| \leqslant \epsilon_{geom}$ translates to the following condition

$$h_S \theta_S \leqslant \epsilon_{geom}.$$

We can then define the geometric variation measure for element T as

$$\eta_T^G = \sum_{S \subset T} h_S \theta_S.$$

Introducing refinement and coarsening parameters $\gamma_R^G, \gamma_C^G > 0$, we obtain Algorithm 6 for geometry-driven space adaptivity.

Algorithm 6 : Geometry-driven adaptivity

```
repeat
  compute η_T^G for all T ∈ T
  let η_max^G = max_{T∈T} η_T^G
  if η_max^G > ε_geom then
    mark elements T with η_T^G > γ_R^G η_max^G
    bisect marked elements d − 1 times
  end if
  mark elements T with η_T^G < γ_C^G η_max^G
  coarsen the marked elements
until mesh is not modified any more
```

4.2.2 Data-driven Adaptivity

In general shape optimization problems may interact with user-specified data in two possible ways:

1. integrating given functions over the boundary or the domain,

2. solving PDEs, whose solution depend on the given functions.

In either case, the accuracy of the computation depends on sufficient resolution of the data. For this reason, we will introduce some error indicators that will help us evaluate local mesh resolution.

In the case of integrals, it is natural to consider adaptive integration to ensure the accuracy of the computation. This is a well-established topic in numerical analysis and is covered in many standard numerical analysis textbooks (e.g. [7]). The main idea here is to try to share the specified error tolerance ϵ_{data} between the elements proportional to their sizes. Given a uniformly continuous function f over a domain (or boundary) U such that $U = \bigcup_{T \in \mathcal{T}} T$, we can compute the exact integral by the following

$$I_f(U) = \sum_{T \in \mathcal{T}} I_f(T) = \sum_{T \in \mathcal{T}} \int_T f(x).$$

On the other hand, the approximate integral on T is given by

$$I_f^h(T) = \sum_{i=1}^{n_Q} w_i f(x_i)$$

where we use quadrature Q with positive weights w_i, and x_i are the quadrature points mapped to element T.

The local resolution criterion is

$$\left|I_f(T) - I_f^h(T)\right| \leqslant \epsilon_{data}\frac{|T|}{|U|}.$$

Since we do not know $I_f(T)$, we replace it with another approximation $\tilde{I}_f^h(T)$ that is evaluated with a higher order quadrature $\tilde{Q}$. So $\tilde{I}_f^h(T)$ gives a better approximation to $I_f(T)$ and we test for

$$\left|\tilde{I}_f^h(T) - I_f^h(T)\right| \leqslant \epsilon_{data}\frac{|T|}{|U|}.$$

We refine T if this condition is not satisfied. The pseudo-code for data-driven adaptivity for integrals is given in Algorithm 7.

Algorithm 7 : Data-driven adaptivity for integrals

```
repeat
  compute η_T^I = |Ĩ_f^h(T) − I_f^h(T)| for all T ∈ 𝒯
  let η^I = Σ_{T∈𝒯} η_T^I
  if η^I > ε_int then
    mark elements T with η_T^I > (|T|/|U|) ε_int
    bisect marked elements d − 1 times
  end if
until mesh is not modified any more
```

In the case of PDEs, one is inclined to consider a posteriori error estimators to guide the adaptivity. This may seem appropriate since we deal with elliptic PDEs for our examples and the results on a posteriori error estimation are well-developed for elliptic PDEs. However it is questionable whether such an off-the-shelf approach would be suitable for shape optimization. There are two major issues:

1. It is possible to go through very "bad" domains in the intermediate stages of the optimization and one may waste unnecessary computational effort by concentrating on recovery of an accurate solution to the PDE in these domains.

2. In shape optimization, we always have an approximation to the domain and no direct clue of the exact domain. In such a case pretending to have the exact domain may be counter-productive. For example, reentrant corners on the polygonal approximation may not really exist on the exact domain and we could be refining in vain.

Another possibility in the PDE case is to have the resolution of the data as the driving criterion for adaptivity. For this we take the interpolation $\Pi_h f$ of the given function f in the current finite element space and check the L^2 error between $\Pi_h f$ and f. Then the test criterion is

$$\|f - \Pi_h f\|_{L^2(U)} \leqslant \epsilon_{data}.$$

If this is not satisfied, we refine the elements with largest error. Let us introduce refinement and coarsening parameters $\gamma_R^P, \gamma_C^P > 0$. Then the pseudo-code for data-driven adaptivity for PDEs is given in Algorithm 8.

Algorithm 8 : Data-driven adaptivity for PDEs

repeat
  compute $\eta_T^P = \|f - \Pi_h f\|^2_{L^2(T)}$ for all $T \in \mathcal{T}$
  let $\eta^P = \sum_{T\in\mathcal{T}} \eta_T^P$
  let $\eta_{max}^P = \max_{T\in\mathcal{T}} \eta_T^P$
  if $\eta^P > \epsilon_{data}$ **then**
    mark elements T with $\eta_T^P > \gamma_R^P \eta_{max}^P$
    bisect marked elements $d-1$ times
  end if
  mark elements T with $\eta_T^P < \gamma_C^P \eta_{max}^P$
  coarsen the marked elements
until mesh is not modified any more

4.3 Topological Changes in 2D

In this section we describe a general-purpose procedure to perform topological changes, such as merging and splitting, for curves in 2d. The method is generic in the sense that it can be used for any curve evolution that requires automatic topological changes as long as *two changes do not occur at same location at the same time*.

Algorithms for topological changes with curves have been introduced before, especially within the context of image segmentation (see [26, 40, 45]). Our method differs from previous work in two points:

1. It utilizes the well-known line sweeping algorithm to guarantee fast detection in $O(n \log n)$ time where n is the number of elements,

2. It incorporates detailed procedures to handle complex scenarios with several topological changes and possible pathological behavior, such as multiple elements intersecting each other.

In particular, the latter are not addressed in [26, 40, 45], which makes previous work more suitable for curves with uniform element size and milder topology change scenarios.

Our method is based on the assumption that we deal with simple closed oriented curves. In this way, we can assign an inside and an outside to a curve. We take the convention that the inside of a counterclockwise oriented curve corresponds to a domain and the normal of the curve points outside. If, for example, we have a curve inside another counterclockwise oriented curve, it has to be oriented clockwise.

Then it corresponds to a hole inside the domain represented by the enclosing curve (see Figure 4.9). This assumption allows us to perform topological changes for very general evolutions without any ambiguity.

4.3.1 Topology-Specific Data Structures

On the algorithmic level, the curve is represented as a polygon that consists of directed line segments, such that there is exactly one incoming and one outgoing line segments or *edge* for each node. These are the *elements*. Then we represent all the curves by a global list of elements. Each element in the list has a start node and an end node. The elements also contain pointers to previous and next neighbors on its curve (see Figure 4.1).

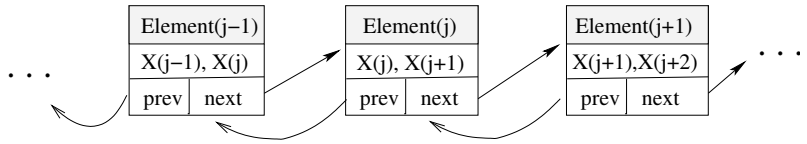


Figure 4.1: Each element stores its start node, end node and pointers to previous and next elements in the curve.

While the list of elements is the central data structure for the method, at certain stages of the algorithm we need to compute the information of *components*, i.e. the information on individual closed curves. This information consists of the component's elements, its signed *area*, its *parent* (the enclosing curve if it is inside one) and its *children* (the immediate curves it encloses if any). See Figure 4.2 illustrating the component structure.

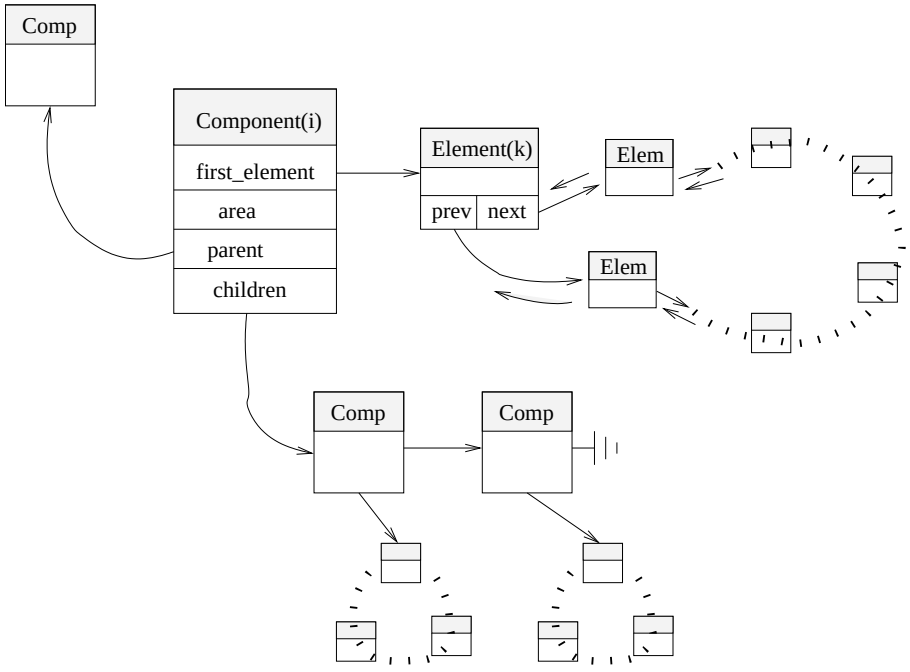


Figure 4.2: The component structure stores the following information of its curve: its signed *area*, a pointer to its *parent*, a pointer to a list of its *children* and a pointer to an arbitrary *first_element* on its curve. The pointer to *first_element* allows to access all elements of its curve via its pointers to *prev* and *next* neighbors.

4.3.2 Topological Changes

The topology algorithm has two phases: detecting the topological events and performing the corresponding topology surgeries.

Detection of topological events is based on the detection of element intersections using the well-known line-sweeping algorithm [23] from computational geometry. Given n elements, this algorithm is guaranteed to have $O((n + I) \log n)$ time complexity where I is the number of intersections. We describe this algorithm in § 4.3.2.2.

Detection of topological events is followed by the corresponding topology surgeries. The topology surgeries consist of a number of steps, where we tune the time step and the local resolution at the intersections, we reconnect the elements and delete the *phantom* components post-reconnection. These operations have linear time complexity.

4.3.2.1 Possible Topological Events

Although the topological events look the same locally at the element intersection level, they may correspond to qualitatively different events at the global level. There are five basic possibilities (see Figure 4.3):

- *merging*: two different components touch each other and merge to create a new component,
- *inside opening*: a curve inside another touches the enclosing curve and they become one curve with a concavity,

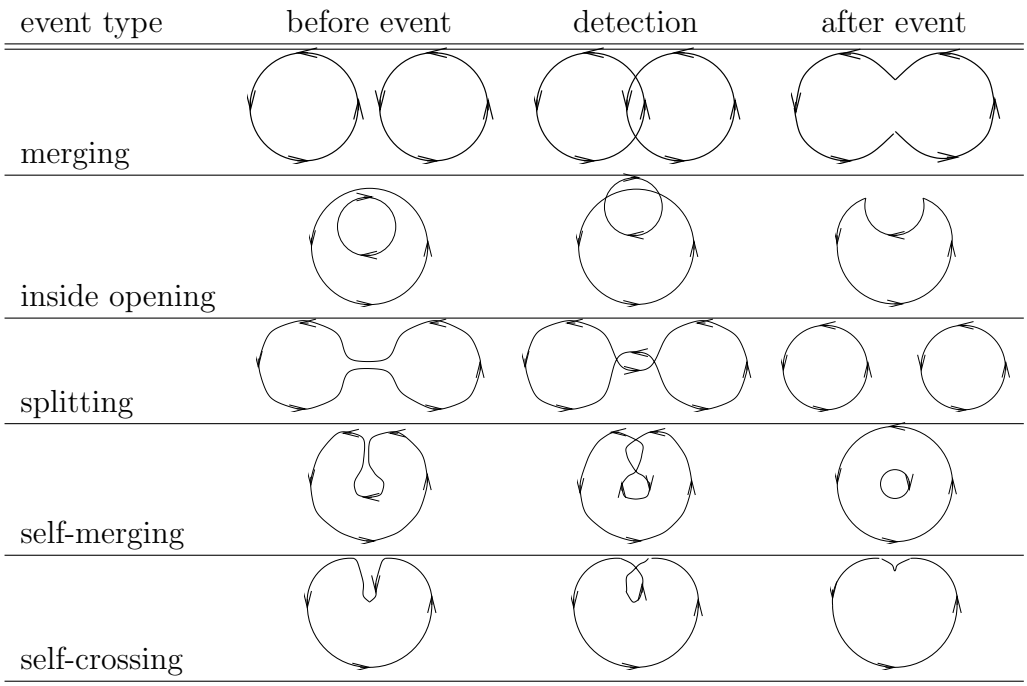


Figure 4.3: We illustrate all basic topological events. The examples show possible configurations right before the event, at the time of detection and after the event with the correct topology.

- *splitting*: the domain enclosed by the curve gets thinner and thinner in a certain region and develops a pinch-off and the curve splits into two,

- *self-merging*: a curve touches itself and this results in another curve inside which basically corresponds to a hole in the domain,

- *self-crossing*: a part of the curve may cross itself and form a loop, which is an anomaly rather than a valid topological event and is thus removed.

4.3.2.2 Detection of Topological Events

Element intersections signal topological events. Note that each valid topological event results in two intersections while a self-crossing yields only one intersection.

A naive algorithm for intersection detection could be devised based on pairwise testing of elements. This would result in $O(n^2)$ time complexity where n is the number of elements. This is in a sense optimal as the running time cannot be less if each element intersects every other element.

On the other hand we can try to exploit the inherent spatial ordering of the elements in plane and in particular the fact that the number of intersections we expect is small with respect to n. For this, we consider an imaginary line sweeping the plane from left to right. This provides some ordering of the elements from left to right. At one instant of the sweeping, the sweep line intersects some of the elements and this gives another ordering for these elements from bottom to top (see Figure 4.4). We can make use of this information to avoid unnecessary intersection checks by testing only the elements in the vicinity of an element with respect to these orderings.

The heart of the line-sweeping algorithm is the choice of the instants when we do the intersection test (after all we cannot really simulate the continuous sweeping of the line). These instants are called the events, specifically line sweep events. They are initially given by the end points of the elements ordered from left to right. We store these events in event queue. As we detect the intersections, we also add them to the event queue in the right order.

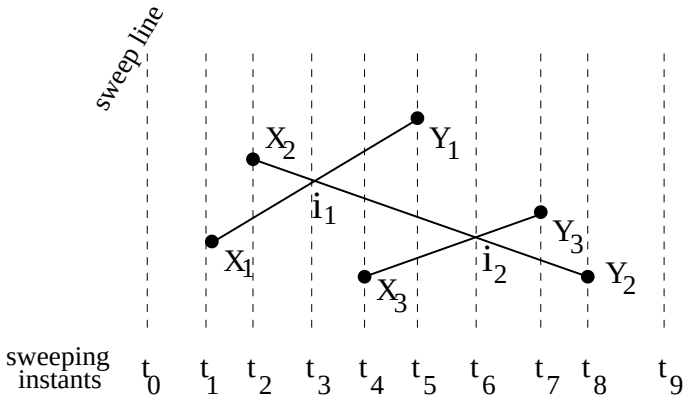
In order to simulate the sweep line, we extract the events from the event queue one by one. For each right (left) end point event, we update the sweep line status to include (exclude) the corresponding element and we check for intersections with the neighbors on sweep line. If there are intersections, we add them to the event queue

as intersection events. The significance of the intersection events is that when the sweep line hits an intersection event, the order of the intersecting elements on the sweep line changes. See Algorithm 9 for a more detailed presentation.

Algorithm 9 : Detect element intersections

```
add end points of all elements to event_queue in order from left to right
while event_queue is not empty do
  remove next event from event_queue
  if it is a left end point event then
    add the element of the end point to sweep_line_status
    check for intersection of element with its neighbors on sweep line
    if there is an intersection then
      add the intersection to event_queue and intersection_list
    end if
  else if it is a right end point event then
    remove the element of the end point from sweep_line_status
    check for intersection of element's neighbors on sweep line
    if there is an intersection then
      add the intersection to event_queue and intersection_list
    end if
  else {it is an intersection event}
    change the order of the intersecting elements in sweep_line_status
    check these elements for intersection with their new neighbors on sweep line
  end if
end while
return intersection_list
```

In this algorithm, the event queue and the line sweep status, which keeps track of the elements intersecting the sweep line, are realized as balanced binary trees. Hence each operation in the algorithm has at most $O(\log n)$ cost resulting in $O(n \log n)$ time complexity (plus $O(I \log n)$ to report the intersections). See [23] for more details on the algorithm and some proofs of correctness. In Figure 4.4, we give an example illustrating the algorithm.



time	event	event queue	sweep line status	detected
t_0	$\{\}$	$\{X_1, X_2, X_3, Y_1, Y_3, Y_2\}$	$\{\}$	$\{\}$
t_1	$\{X_1\}$	$\{X_2, X_3, Y_1, Y_3, Y_2\}$	$\{(X_1, Y_1)\}$	$\{\}$
t_2	$\{X_2\}$	$\{X_3, Y_1, Y_3, Y_2\}$	$\{(X_1, Y_1), (X_2, Y_2)\}$	$\{\}$
t_2	$\{X_2\}$	$\{i_1, X_3, Y_1, Y_3, Y_2\}$	$\{(X_1, Y_1), (X_2, Y_2)\}$	$\{i_1\}$
t_3	$\{i_1\}$	$\{X_3, Y_1, Y_3, Y_2\}$	$\{(X_2, Y_2), (X_1, Y_1)\}$	$\{i_1\}$
t_4	$\{X_3\}$	$\{Y_1, Y_3, Y_1\}$	$\{(X_3, Y_3), (X_2, Y_2), (X_1, Y_1)\}$	$\{i_1\}$
t_4	$\{X_3\}$	$\{Y_1, i_2, Y_3, Y_2\}$	$\{(X_3, Y_3), (X_2, Y_2), (X_1, Y_1)\}$	$\{i_1, i_2\}$
t_5	$\{Y_1\}$	$\{i_2, Y_3, Y_2\}$	$\{(X_3, Y_3), (X_2, Y_2)\}$	$\{i_1, i_2\}$
t_6	$\{i_2\}$	$\{Y_3, Y_2\}$	$\{(X_2, Y_2), (X_3, Y_3)\}$	$\{i_1, i_2\}$
t_7	$\{Y_3\}$	$\{Y_2\}$	$\{(X_2, Y_2)\}$	$\{i_1, i_2\}$
t_8	$\{Y_2\}$	$\{\}$	$\{\}$	$\{i_1, i_2\}$
t_9	$\{\}$	$\{\}$	$\{\}$	$\{i_1, i_2\}$

Figure 4.4: We illustrate the intersection detection algorithm on a simple example. The events including the intersections take place at instants $t_i, i = 1, \ldots, 8$. We initialize the *event queue* with all end points. Then we extract each event from *event queue* and process it according to Algorithm 9. We update the *sweep line status* at each event. When we detect intersections, we add them to *event queue* and the list of *detected*.

4.3.2.3 Topology Surgery

Once we have detected element intersections, we should decide how to proceed with the topology surgery, that is the correction of the curve for the new topology. The topology surgery consists of the following steps:

1. take the curve back to the previous iteration,
2. adjust the time step for surgery,
3. move the curve forward with adjusted time step,
4. refine intersections before surgery,
5. reconnect intersecting elements,
6. compute component information,
7. remove phantom components.

Now let's take a closer look at some of these steps.

Adjusting time step for surgery

Given a set of pending topological events, we may want to resolve them at different levels. For example, for a physical simulation, we may want to realize one topological event at a time. Then we make use of the fact that a single valid topological event is signalled by two intersections and we try to reduce the time step to capture this.

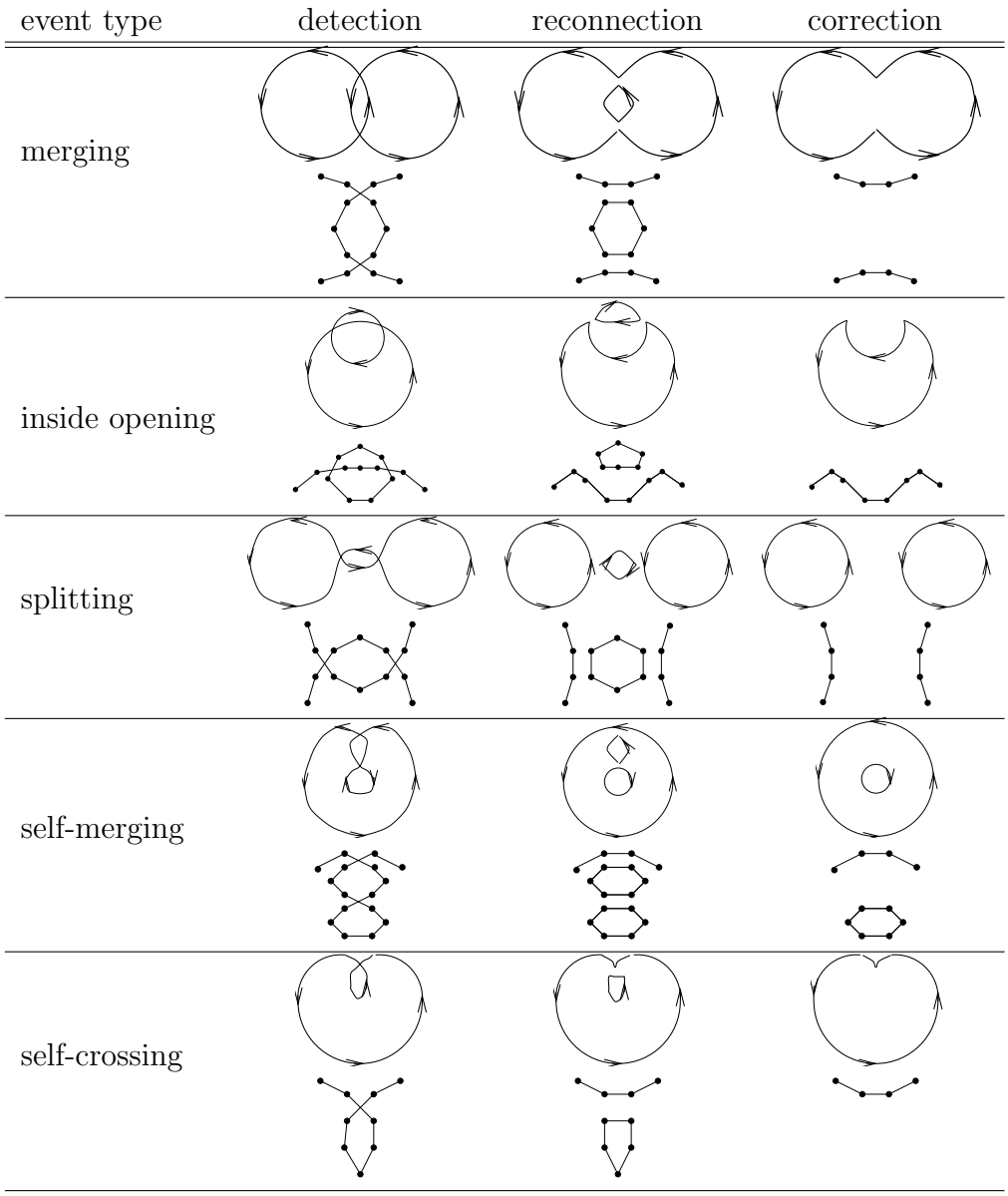


Figure 4.5: We illustrate the three main stages of topology surgery at the continuous and discrete levels. These are 1)detection of intersections, 2)reconnection of the intersecting elements, 3)correction by deletion of phantom components.

We may also want to have a better resolution of an individual topological event, in particular, we may want to capture the moment of touching. In this case we try to reduce the time step to capture the time when only one node of each curve has passed through the other, in other words, the intersections take place only at neighboring elements for each curve. Still we should point out that it is not always possible to catch or resolve a single first event. In that case we process the first events that we obtain at the reduced time step.

On the other hand, for a shape optimization application like image segmentation, we may simply want to do the surgery for all pending events simultaneously. Hence we do not adjust the time step and we take it as it is. We refer to Algorithm 10 for more details and Figure 4.6 for an illustration of all these cases.

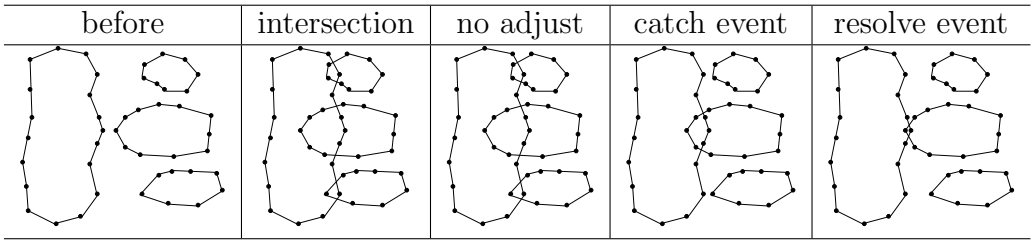


Figure 4.6: We illustrate different time step adjustment strategies with this example. A large time step results in simultaneous collision of the three smaller curves with the large curve. If we choose to do no adjustment, we do not change the time step. If we choose to catch the first topological event, we reduce the time step to have only two element intersections. If we choose to resolve the first event, we reduce the time step to have only two intersections at neighboring locations.

Refining intersections before surgery

In order to do the reconnections correctly, some elements need to be refined. This also helps to increase the quality of reconnections for robustness. The refine-

Algorithm 10 : Adjust timestep for surgery

if *timestep_adjustment* = NONE **then**
  return *intersection_list* and *timestep* unchanged
end if
$k = 0$
$a = min\_timestep$
$b = timestep$
while $k < max\_iteration$ **do**
  $timestep = a + (b - a)/2$
  compute new location of curve with *timestep* and current velocity
  intersection_list = the intersections on new location
  if size of *intersection_list* > 2 **then**
    $b = timestep$
  else if size of *intersection_list* $= 0$ **then**
    $a = timestep$
  else if size of *intersection_list* $= 1$ **then**
    {it is a self-crossing}
    exit while loop
  else {we have size of *intersection_list* $= 2$}
    get the two intersections from *intersection_list*
    if *timestep_adjustment* = RESOLVE_FIRST_EVENT and intersections do not take place at neighboring elements **then**
      b = *timestep*
    else {either *timestep_adjustment* = CATCH_FIRST_EVENT or intersections take place at neighboring elements}
      exit while loop
    end if
  end if
  $k = k + 1$
end while
return *intersection_list* and *timestep*

ment has two phases:

- Refine the elements with multiple intersections, so that we have a list of disjoint intersections, i.e. no element is intersected more than once (see Figure 4.7 for an illustration),

- Take each intersection and bring the elements of intersections to comparable size. We do this to avoid cases where we may have a very long element intersected by a very short element (see Figure 4.7).

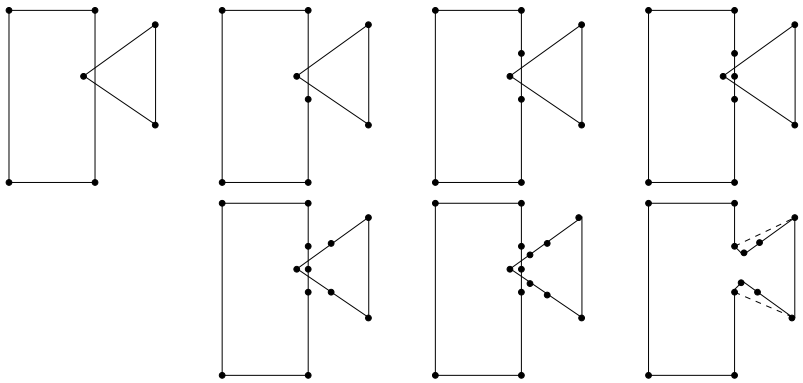


Figure 4.7: We illustrate two phases of intersection refinement with this example. Initially we cannot reconnect elements because one element is intersected by two elements. So we refine the larger element. In the second phase we refine the intersecting elements until they are of comparable size. The dotted lines in the last figure show the result of the reconnection without executing the second phase.

It is also possible to enforce a topology resolution scale at the second phase by refining the elements down to a prescribed size. See Algorithm 11 for the pseudocode.

Reconnecting intersecting elements

Algorithm 11 : Refine intersections before surgery

compute *count* of intersections for each element in *intersection_list*
while there are elements with *count* > 1 **do**
  refine these elements
  compute the intersections of the new elements
  recompute *count* of intersections for new elements
end while
for all *intersections* in *intersection_list* **do**
  if both elements of the *intersection* are not of comparable size **then**
    refine the larger until it compares to the smaller
    update the *intersection*
  end if
  if a *topology resolution scale* is imposed **then**
    refine both elements of the intersection to resolution scale
    update the *intersection*
  end if
end for

Given two directed line segments that are intersecting, reconnection simply consists of interchanging their destination nodes. Hence upon reconnection, the end point of the first line segment becomes the end point of the second element and vice versa (see Figure 4.8).

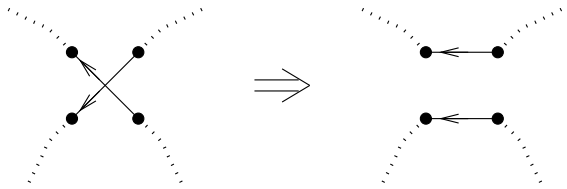


Figure 4.8: This figure shows how the intersecting elements are reconnected. The destination node of one element becomes the destination node of the other.

Computing component information

In order to finalize the topology surgery correctly, we need to compute certain information about the components in the domain. These are the signed area of

each component, the constituent elements of each component and the component hierarchy, that is the parent and children of each component. Signed area encodes both area and orientation information of the curve. See Figure 4.9 for an example of component hierarchy encoding a given layout of curves.

Note that we can compute the area of a domain Ω enclosed by a polygonal curve Γ from the contributions of individual elements T_i as follows:

$$Area = \int_{\Omega} dx = \frac{1}{2}\int_{\Omega} \mathrm{div} x dx = \frac{1}{2}\int_{\Gamma} x \cdot \nu dS = \frac{1}{2}\sum_{i}\int_{T_i} x \cdot \nu dS. \qquad (4.1)$$

We can make use of this identity, along with the element connectivity information, to compute the components as in Algorithm 12. Then given the component list, we can compute the component hierarchy with Algorithm 13.

Algorithm 12 : Compute components

```
add all the elements to an element_list
while element_list is not empty do
  remove first_element from element_list
  create a new component and assign first_element to component
  add area contribution of first_element to component
  add component to component_list
  current_element ⇐ next neighbor of first_element
  repeat
    add area contribution of current_element to component
    remove current_element from element_list
    current_element ⇐ next neighbor of current_element
  until current_element = first_element
end while
return component_list
```

Algorithm 13 has $O(kn)$ time complexity where k is the number of components and n is the number of elements. It is possible to reduce the factor k further with a line sweeping strategy. Another strategy would be to compute the component

Algorithm 13 : Compute component hierarchy

sort *component_list* with respect to decreasing area
for each *component* taken from *component_list* in order **do**
  find the next smallest component enclosing current *component*
  if there is such a component **then**
    call it *parent* and update current *component*'s parent information
    add current *component* to *parent*'s children list
  else
    add current *component* to *outermost_components_list*
  end if
end for

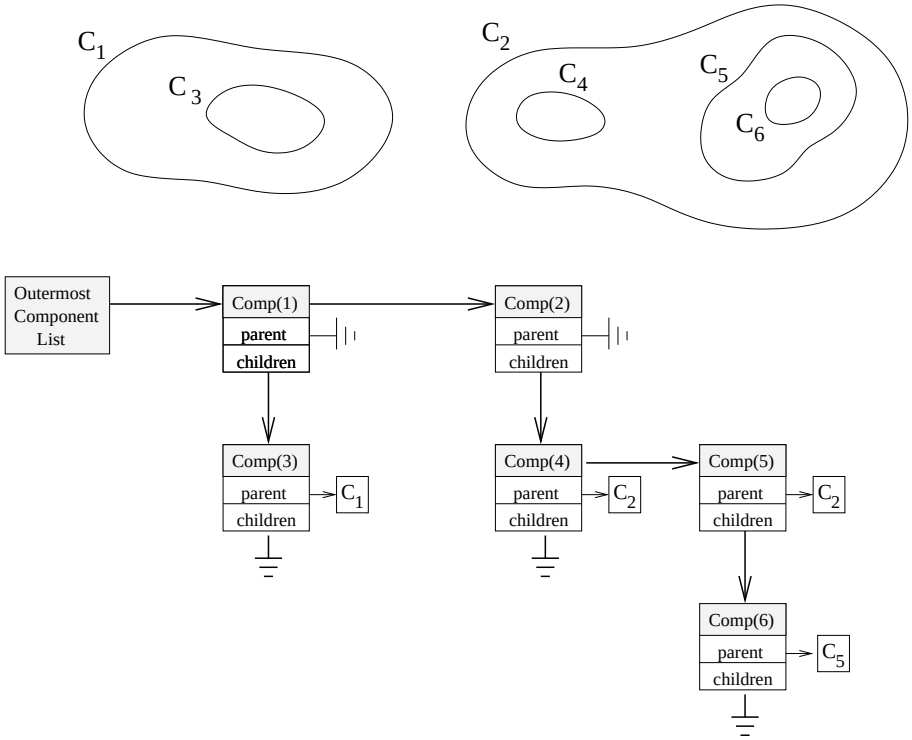


Figure 4.9: Example of some curves with the corresponding component hierarchy

hierarchy at the beginning and to update it as topological events take place.

Removing phantom components

Reconnecting the intersecting elements creates the new components with the right topology. But it also creates some extra components that we call *phantom components* (see Figure 4.5). These are artifacts of the surgery and need to be removed. Note that we can identify them easily as their orientation will be incompatible with the orientations of the valid components. Accordingly the following key ideas form the basis of our correction algorithm:

- All the components that are not inside another component should have the same orientation, which we call *outermost orientation*.

- A component inside another component should have orientation opposite to that of the enclosing component. We should point out that the algorithm yields the right topology as long as simultaneous topological events occur at separate locations. If they occur at the same location, we may get different results depending on the discrete realization of the event. An example of this is given by a curve containing a curve that contains another curve, such that all these curves touch each other at the same point at the same time. To handle these correctly, we need to introduce additional assumptions to decide the correct behavior.

  See Algorithms 14, 15, 16 for the pseudocode.

Algorithm 14 : Remove phantom components

```
for each component in outermost_components_list do
  if component's orientation ≠ outermost_orientation then
    remove_component_and_children( component )
  else
    for each child in component's children list do
      enforce_orientation_compatibility( child )
    end for
  end if
end for
```

Algorithm 15 : enforce_orientation_compatibility(*component*)

```
if component and its parent have the same orientation then
  remove_component_and_children( component )
else
  for each child in component's children list do
    enforce_orientation_compatibility( child )
  end for
end if
```

Algorithm 16 : remove_component_and_children(*component*)

```
for each child in component's children list do
  remove_component_and_children( child )
end for
remove component from parent's children list
delete all elements corresponding to component
```

Chapter 5

Numerical Experiments

In this chapter we apply the results of the previous chapters to two major shape energies used for image segmentation. These are the minimal surface model (1.6) and the Mumford-Shah model (1.12). For these we implement the discrete gradient flows of Chapter 3 using the finite element toolbox ALBERTA [60].

For each model we study the L^2 flow and a weighted H^1 flow. The weighted H^1 flow is based on the second shape derivative and therefore corresponds to an inexact Newton's method. Note that given the first shape derivative $dJ(\Omega;\cdot)$ and the second shape derivative $d^2J(\Omega;\cdot,\cdot)$ of $J(\Omega)$, we can compute the Newton's descent direction by solving

$$d^2J(\Omega;V,W) = -dJ(\Omega;W), \qquad \forall W \in B(\Gamma)$$

if $d^2J(\Omega;\cdot,\cdot)$ is positive definite. $B(\Gamma)$ is the Hilbert space induced by the scalar product $b_N(V,W) = d^2J(\Omega;V,W)$. Even if $d^2J(\Omega;\cdot,\cdot)$ is not positive definite; it is often possible to introduce a positive definite bilinear form $b(\cdot,\cdot)$ by modifying $d^2J(\Omega;\cdot,\cdot)$. In practice we see that this still preserves in part the favorable convergence properties of the Newton's method. Following these ideas we introduce the weighted H^1 flows for both (1.6) and (1.12).

We compute the solutions given by these flows for a number of synthetic and real examples. The favorable stability properties of the H^1 flow allows us to use

the explicit scheme (3.53) for (1.6), whereas we use the implicit scheme (3.54) for the other experiments. For purposes of comparison, we choose the computational domain for the synthetic experiments to be $[-1,1]^2$ in 2d and $[-1,1]^3$ in 3d. The range of the image intensity function is $[0,1]$. For experiments with real images we map the shortest side of the image to $[-1,1]$ and scale the other sides keeping the overall ratio fixed. We also scale the range of the image to $[0,1]$ from the usual values of $[0,255]$. The synthetic images are created to represent a sample of critical features that object boundaries in real images may comprise, such as sharp corners, cusps and concavities.

5.1 The Minimal Surface Model

In this section we will use the minimal surface energy

$$J(\Gamma) = \gamma_0 \int_\Gamma H(x) dS + \gamma \int_\Omega H(x) dx \tag{5.1}$$

where the edge indicator function H, given by

$$H(x) = \frac{1}{1 + \frac{|\nabla I(x)|^2}{\lambda^2}},$$

is used to locate boundaries of objects in 2d and 3d images. We call the parameter $\lambda > 0$ the edge strength. Recall that the first shape derivative is given by

$$dJ(\Omega; V) = \int_\Gamma \Big((\gamma_0 \kappa + \gamma) H(x) + \gamma_0 \partial_\nu H(x) \Big) V dS,$$

whereas the second shape derivative is given by

$$\begin{aligned} d^2 J(\Gamma; V, W) &= \int_\Gamma \gamma_0 H \nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \int_\Gamma \Big(\gamma_0 \partial_{\nu\nu} H + (2\gamma_0 \kappa + \gamma) \partial_\nu H + (\gamma_0 \kappa^2 - \gamma_0 \sum \kappa_i^2 + 2\gamma\kappa) H \Big) V W dS. \end{aligned}$$

Based on this we introduce a weighted H^1 scalar product

$$\begin{aligned}\langle V, W\rangle_{H^1} &= \int_\Gamma \gamma_0 H \nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \int_\Gamma \left(\gamma_0 \partial_{\nu\nu} H + (2\gamma_0\kappa + \gamma)\partial_\nu H + 2\gamma\kappa H\right)_+ VW dS.\end{aligned}$$

where $(\cdot) = \max(\cdot, \epsilon)$ and we choose $\epsilon = 100$ (because using lower values deteriorates the conditioning of the matrix $A_\alpha + M_\beta$ introduced in §3.5; one can easily get the intuition for this by examining the coefficient matrix corresponding to the discretization of the simple differential operator $Ly = ay'' + by$ where a,b are constants). As this is positive definite, we can set $b(V, W) = \langle V, W\rangle_{H^1}$ and solve the system (3.4) to obtain an inexact Newton's descent direction. We will illustrate this H^1 flow and the L^2 flow with synthetic and real examples.

There are a few practical issues regarding the implementation. The first is time step selection, which we handle using the line search procedure described in Algorithm 2. The parameters α, β of line search are set to be 0.25 and 0.75 respectively. $\tau_{min} = 10^{-4}, 10^{-2}$ and $\tau_{max} = 10^{-2}, 1$ for L^2 and H^1 flows respectively. Another important issue is the stopping criterion. There are two possibilities: tracking the change in energy or tracking the shape derivative. We use the second approach for most of the experiments. Then the stopping criterion is:

$$dJ(\Omega_k; V_k) \leqslant tol_{rel} \cdot dJ(\Omega_0; V_0) + tol. \tag{5.2}$$

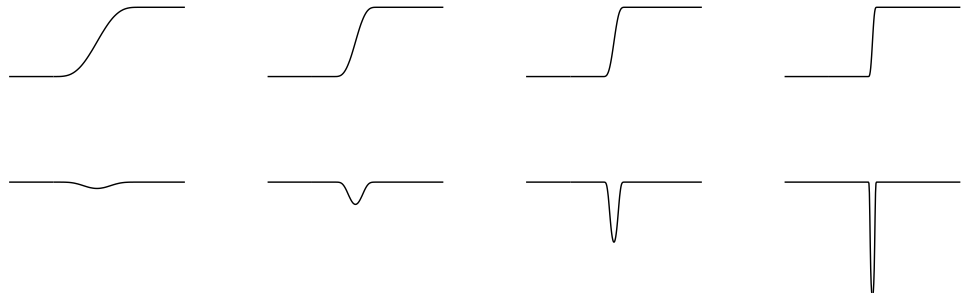
For almost all the experiments, we chose to set $tol_{rel} = 0$ and specified suitable tol values.

Finally, in order to gain efficiency and accuracy simultaneously, we used the geometry-driven space adaptivity and data-driven space adaptivity procedures de-

scribed in Algorithms 6 and 7 respectively. The parameters for these were $\epsilon_{geom} = 0.02$, $\gamma_R^G = 0.7$, $\gamma_C^G = 0.3$, $\epsilon_{int} = 0.2$. The time step control procedure described in Algorithm 3 was also used with $\epsilon_\tau = 0.8$. However we turned off space adaptivity in the synthetic experiments as we wanted to study the behavior of the model (5.1) keeping the computational procedure as plain as possible.

5.1.1 Evaluating the Model

The energy (5.1) depends on two parameters γ and λ ($\gamma_0 = 1$ for most of the experiments). We examine the behavior of the model with respect to changing γ and λ on smooth synthetic images in §5.1.1.1 and §5.1.1.2 respectively. We also examine the effect of the image characteristics on model (5.1) in §5.1.1.3. One critical variable is the edge width, ε_{edge}, which is the scale of the transition between the image intensity values across the edge. For a discontinuity at the image we have $\varepsilon_{edge} = 0$. But to be able to compute the first and second shape derivatives we need the first and second derivative of $H(x)$, hence up to third derivatives of the image. For this reason, we need to have smooth transitions at the edges and ε_{edge} specifies the scale of the transition. In addition to edge width, we examine the effect of a nonuniform background in the image for computing the model (5.1). This includes varying the image intensity linearly and adding oscillating texture patterns.



$\varepsilon_{edge} = 0.1$ $\varepsilon_{edge} = 0.05$ $\varepsilon_{edge} = 0.025$ $\varepsilon_{edge} = 0.01$

Figure 5.1: The effect of varying the edge width parameter ε_{edge} illustrated with a 1d image. The top row shows the image intensity function $I(x)$ and the bottom row shows the edge indicator function $H(x)$.

5.1.1.1 The Effect of Edge Width and Edge Strength

We present experiments demonstrating the effect of varying ε_{edge} and λ. The synthetic images used are given in Figure 5.2. We start each experiment with a circular curve of radius one centered at the origin. The initial energy of the curve is $J_0 = 6.283$. We use 256 nodes to represent the curve. The termination parameter *tol* for the L^2 and the H^1 flows is set to be 10^{-4}. For each experiment we list the iteration number k and final energy J in the corresponding figures.

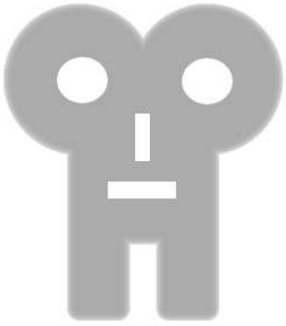
In the first set of experiments we set $\lambda = 50$ and $\gamma = 0$. We vary edge width as $\varepsilon_{edge} = 0.01, 0.025, 0.05, 0.1$. The effect of this on the image function $I(x)$ and edge indicator function $H(x)$ is illustrated in Figure 5.1. The sequence of synthetic images created in this way is shown in Figure 5.2.

We observe that both L^2 flow and H^1 flow perform well for $\varepsilon_{edge} = 0.025, 0.05$ as shown in Figure 5.3. For $\varepsilon_{edge} = 0.1$, they miss the object boundary. But setting $\lambda = 10$ increases the emphasis of the image derivatives in $H(x)$ and edges are perceived more strongly. This corrects the behavior as shown in Figure 5.4.

$\varepsilon_{edge} = 0.01$

$\varepsilon_{edge} = 0.025$

$\varepsilon_{edge} = 0.05$



$\varepsilon_{edge} = 0.1$

Figure 5.2: The sequence of synthetic images created by varying the edge width at the boundary, i.e. the width of the transition region.

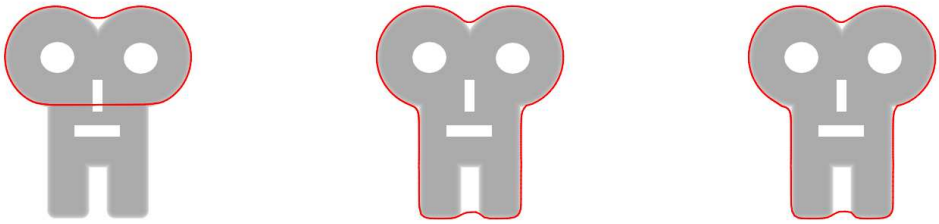
L^2 flow	L^2 flow	H^1 flow	H^1 flow
$\varepsilon_{edge} = 0.025$	$\varepsilon_{edge} = 0.05$	$\varepsilon_{edge} = 0.025$	$\varepsilon_{edge} = 0.05$
$k = 1026$	$k = 361$	$k = 693$	$k = 298$
$J = 0.488$	$J = 1.161$	$J = 0.509$	$J = 1.153$

Figure 5.3: The segmentation results obtained with the L^2 and H^1 flows for different edge widths. Both terminate successfully, but they miss the concavity.

Choosing to have sharp edges by setting $\varepsilon_{edge} = 0.01$, on the other hand, proves to be challenging for the method. Both L^2 and H^1 flows miss the boundary. Edge width is very small and the image derivatives are very sharp at the boundary. We set $\lambda = 400$ to decrease the emphasis of the image derivatives in $H(x)$. This does not address the problem completely as the element size is not small enough to resolve edge transition region. We see that using 1024 nodes instead of 256 nodes solves this problem by quadrupling the computational cost. So we choose to switch on space adaptivity. In this way we start with 32 nodes and gradually increase to $\approx$ 700 nodes and the results are satisfactory as shown in Figure 5.5.

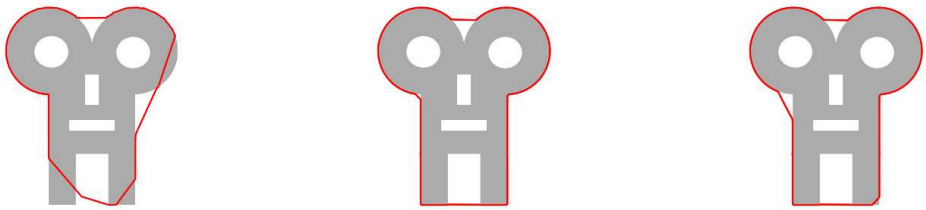
In the second set of experiments, we vary the edge strength λ by setting $\lambda = 10, 25, 50, 100$. It is easy to see that for $\nabla I(x) \neq 0$ fixed, $H(x)$ approaches 1 if λ gets large. Similarly $H(x)$ approaches 0 if λ gets small. This is illustrated in Figure 5.6. For the experiments, we fix $\varepsilon_{edge} = 0.05$ and $\gamma = 0$.

The method performs well for $\lambda = 25, 50$ as shown in Figure 5.7. On the other hand, setting $\lambda = 100$ decreases the edge strength too much and the L^2 flow



L^2 flow	L^2 flow	H^1 flow
$k = 3813$	$k = 280$	$k = 66$
$J = 1.949$	$J = 0.295$	$J = 0.291$

Figure 5.4: The results of the segmentation for $\varepsilon_{edge} = 0.1$ Both L^2 and H^1 flows miss the boundary of the object when $\lambda = 50$. By setting $\lambda = 10$, we obtain the correct segmentation as shown in the second and the third images.



L^2 flow	L^2 flow	H^1 flow
$k = 4360$	$k = 16479$	$k = 4952$
$J = 1.409$	$J = 1.993$	$J = 2.072$

Figure 5.5: The results of the segmentation for $\varepsilon_{edge} = 0.01$. Both L^2 and H^1 flows fail to detect the boundary of the object when $\lambda = 50$. The edge indicator H varies too sharply in the narrow transition region. By setting $\lambda = 400$ and turning space adaptivity on to ensure sufficient curve resolution, we obtain the correct segmentation as shown in the second and the third images.

$\lambda = 10$	$\lambda = 25$	$\lambda = 50$	$\lambda = 100$

Figure 5.6: We fix $\varepsilon_{edge} = 0.05$ and vary λ. The effect of this on the edge indicator function $H(x)$ in 1d is as shown in the sequence.

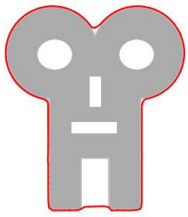
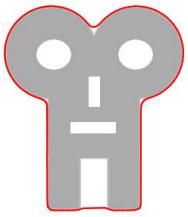
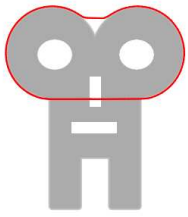
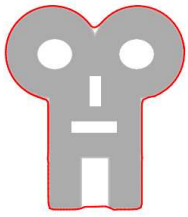
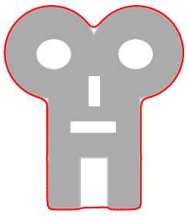
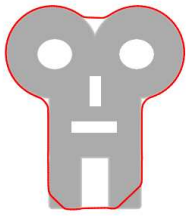
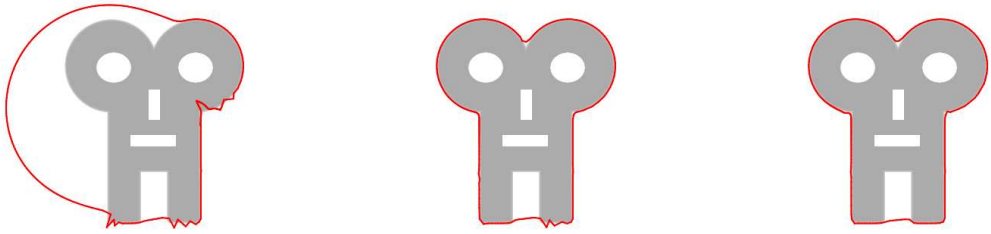
$\lambda = 25$	$\lambda = 50$	$\lambda = 100$
		
$k = 245$ $J = 0.449$	$k = 361$ $J = 1.161$	$k = 3216$ $J = 1.855$
		
$k = 382$ $J = 0.467$	$k = 298$ $J = 1.153$	$k = 671$ $J = 2.475$

Figure 5.7: The segmentation results obtained by varying the values of λ. The top row shows the results for the L^2 flow. The bottom row shows the results for the H^1 flow.

misses the object boundary and stops at a local minimum. The H^1 flow still gives a reasonable result. Similarly for $\lambda = 10$, the edge strength is too large and it causes large velocities in the transition region. This creates unstable behavior with the L^2 flow. Taking smaller time steps corrects this problem. Meanwhile the H^1 flow captures the boundary successfully with the time steps unchanged (see Figure 5.8).



L^2 flow
$k = 433$

L^2 flow
$k = 635$
$J = 0.167$

H^1 flow
$k = 118$
$J = 0.191$

Figure 5.8: The L^2 flow fails for $\lambda = 10$ as the variation in the edge indicator function $H(x)$ is too sharp at the boundary of the object. The H^1 flow, on the other hand, captures the object boundary successfully. Taking smaller time steps with L^2 flow corrects the problem as shown in the middle figure.

5.1.1.2 The Effect of the Domain Term

In the experiments with $\gamma = 0$, we observe difficulty in detecting the concavities of the object. Caselles et al. propose the extra force generated by choosing nonzero γ in the original paper for geodesic active contours [18]. Note that $\gamma > 0$ provides an additional force pushing the curve inwards whereas $\gamma < 0$ pushes the curve outwards.

We examine the effect of changing the magnitude of γ by a set of experiments with $\gamma = 0, 10, 20, 30$. For these experiments we set $\varepsilon_{edge} = 0.05$. The initial curve is a circle of radius one with the object at the center. We set $tol = 10^{-4}$ and 5×10^{-4} for L^2 and H^1 flows respectively. The number of nodes is 512. The main goal in this set of experiments is to test the success in detecting the concavity. So to ensure that we have enough resolution in the region of concavity, we turn on the mesh smoothing procedure described in Algorithm 4 with $n_{MS} = 1$. This equidistributes the nodes throughout the evolution.

The experiments complete successfully for $\gamma = 0, 10, 20$ as shown in Figure 5.9. In particular with $\gamma = 20$, we detect the concavity completely. On the other hand we observe $\gamma = 30$ to be too strong: both L^2 and H^1 flows miss the boundary (Figure 5.10). This in fact is caused by the velocity being too large. So reducing the time step will solve this problem. However this introduces another variable that needs to be adjusted for the experimentation. Instead we find that it is more practical to reduce the values of the paramters γ_0 and γ while keeping their ratio the same. Using $\gamma_0 = 0.2$ and $\gamma = 6$ instead of $\gamma_0 = 1$ and $\gamma = 30$, we capture the boundary successfully. Accordingly, in our experiments with real images, we prefer to keep $\gamma = 1$ and adjust the value of γ_0. In this way we do not have to readjust the time steps, because the order of magnitude is the same for all the velocities specified this way.

5.1.1.3 The Effect of Nonuniform Background

In order to gain insight about the behavior of the method on images with nonuniform background, we execute two sets of experiments. First we add a linearly-changing intensity to the background. So given the original intensity value $I(x_1, x_2)$ at a background point (x_1, x_2), the new intensity value is

$$\tilde{I}(x_1, x_2) = I(x_1, x_2) + \frac{m}{4}(2 + x_1 + x_2).$$

m is a parameter specifying the rate of change. Then we test the effect of adding a simple texture pattern. We add the oscillation given by

$$z(x_1, x_2) = A \cos\left(\frac{\pi x_1}{N\varepsilon_{edge}}\right) \tag{5.3}$$

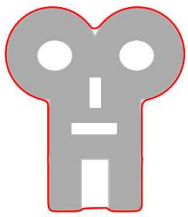
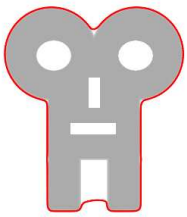
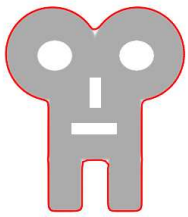
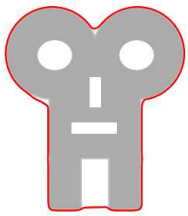
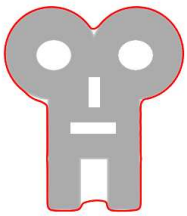
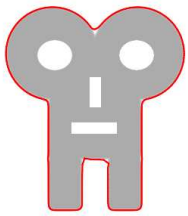
$\gamma = 0,\ J_0 = 5.655$	$\gamma = 10,\ J_0 = 31.101$	$\gamma = 20,\ J_0 = 56.547$
		
$k = 272$ $J = 0.454$	$k = 97$ $J = 8.819$	$k = 179$ $J = 16.337$
		
$k = 203$ $J = 0.454$	$k = 62$ $J = 8.877$	$k = 129$ $J = 16.343$

Figure 5.9: The segmentation results for different values of γ. The top and bottom rows show the results obtained by the L^2 and H^1 flows respectively.

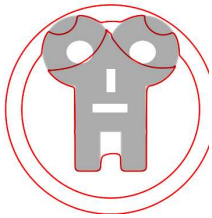
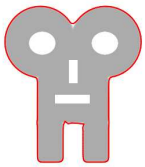
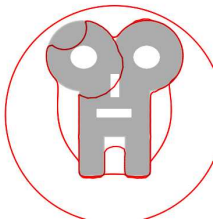
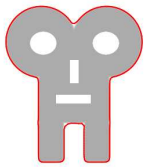
			
L^2 flow $\gamma_0 = 1,\ \gamma = 30$ $k = 4, 15, 205$	L^2 flow $\gamma_0 = 0.2,\ \gamma = 6$ $k = 382$	H^1 flow $\gamma_0 = 1,\ \gamma = 30$ $k = 3, 20, 47$	H^1 $\gamma_0 = 0.2,\ \gamma = 6$ $k = 442$

Figure 5.10: The curves miss the boundary for $\gamma = 30$. We correct this by reducing the weights of the integrals, but keeping their ratio the same. This is effectively the same as taking smaller time steps. The initial energy is $J_0 = 16.399$ in this case, and is reduced to 4.845 by the L^2 flow and to 4.851 by the H^1 flow.

to the background. We examine the impact of changing the amplitude A and the frequency $\propto \frac{1}{N}$.

Note that for a circular object with a given edge width ε_{edge}, the image derivative at the transition region will be proportional to $-\frac{1}{\varepsilon_{edge}}(\cos\theta, \sin\theta)$. For $\theta = \frac{\pi}{4}$ (in the direction of the top right corner) this is $-\frac{1}{\varepsilon_{edge}}(1,1)$, whereas for $\theta = \frac{5\pi}{4}$ (in the direction of the bottom left corner) it is $\frac{1}{\varepsilon_{edge}}(1,1)$. So if we add $\frac{m}{4}(1,1)$ to these, it will weaken the effect of the image derivative at $\theta = \frac{\pi}{4}$ and strengthen its effect at $\theta = \frac{\pi}{4}$. Fixing $\varepsilon_{edge} = 0.05$, $\lambda = 25$ and $\gamma = 0$, we set $m = 1$ and test the scenario where the background intensity changes linearly from the minimum value 0 at $(-1,-1)$ to the maximum value 1 at $(1,1)$. We terminate when the energy change is below 10^{-6}. We observe that qualitatively the result of this experiment is the same as having $m = 0$. However as we increase the value of m, we observe the expected behavior (see Figure 5.11). For $m = 10$, the result is still reasonable. For $m = 20$, the methods fail to capture the concavities with edges facing the direction $(1,1)$.

Next we examine the effect of adding the texture given by (5.3). We use the same parameters in (5.1) as the previous experiment and set $tol = 5 \times 10^{-4}$. If we choose $A = 1$ and $N = 2$, this results in a pattern where the period of the oscillation is equal to $4\varepsilon_{edge}$ and this creates background derivatives comparable to the edge derivatives in magnitude. Naturally we can expect this to create difficulties in the segmentation. Indeed the curves keep getting stuck at the waves as shown in Figure 5.12 and both L^2 and H^1 flows stop at local minima. However decreasing the amplitude to $A = 0.3$ or increasing N to 4, we decrease the effect of the derivative

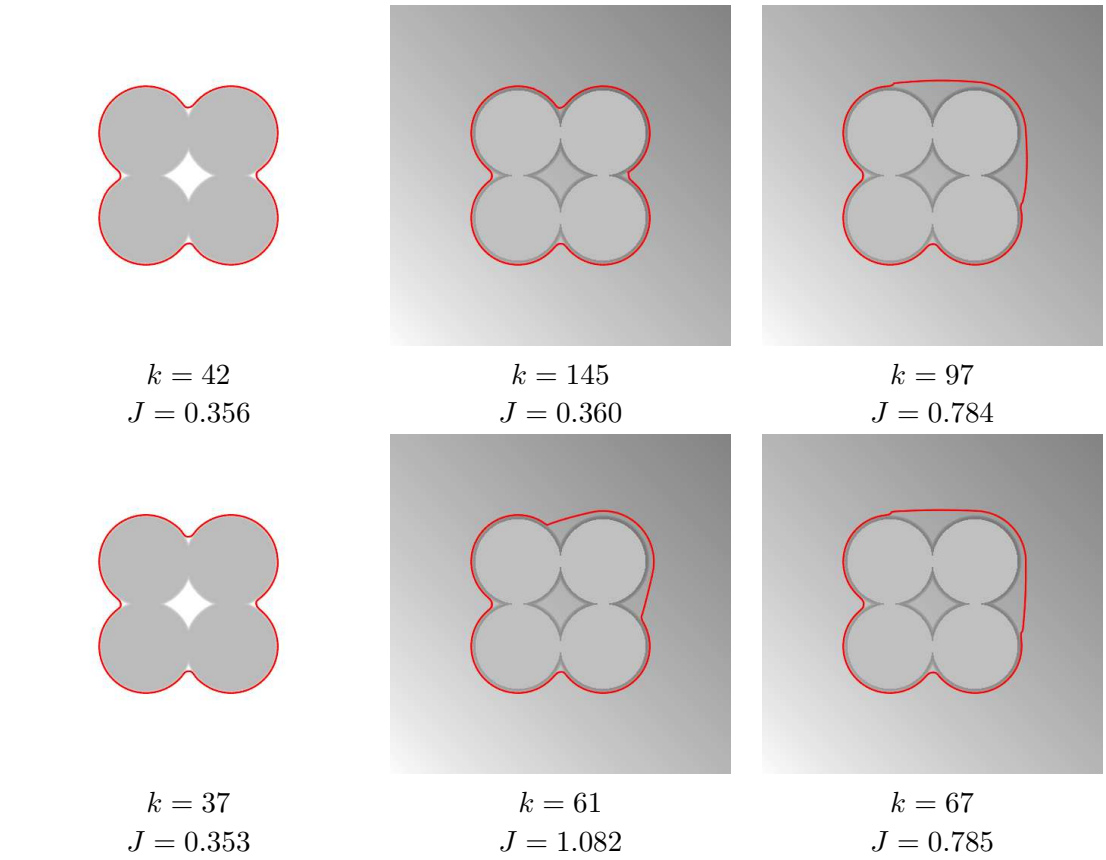


Figure 5.11: The segmentation results for different values of the parameter m. The background intensity changes from 0 at the bottom left corner to m at the top right corner. The top and the bottom rows show the results for the L^2 and H^1 flows respectively. Note that the change of the background intensity in the images does not really reflect m. We use the same generic background for the four images on the right, for purposes of illustration only.

of the texture and we capture the boundary of the object completely.

5.1.1.4 Conclusions

Our experiments give us certain insight about the behavior of model (5.1). We use this to guide our experimentation with real images. In particular they hint at the need for a certain degree of preprocessing. The main conclusions from the experiments with synthetic data are as follows:

- The method performs better if the minima in $H(x)$ (corresponding to the edges) are relatively well-behaved and are not too shallow or too steep. This can be adjusted through λ.

- The method converges faster if edge width is larger. So very sharp edges in images degrade the performance. It is necessary to apply some blurring or smoothing to the images before using the method.

- To be able to detect concavities, it is necessary to choose a nonzero value for γ (the weight of the domain term in (5.1)). A more practical way to do this is to set $\gamma = 1$ and adjust γ_0, the weight of the boundary term, in order to get the desired behavior. In this way, we do not need to adjust the time step scale as an additional parameter.

- Existence of varying patterns in the background make detection more difficult, especially if their derivatives are comparable in magnitude to image derivatives at edges. One may need to apply some preprocessing that wipes out the

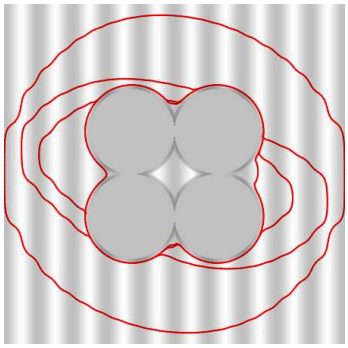
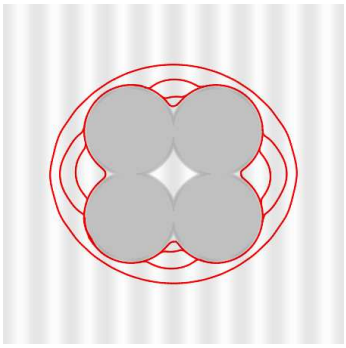
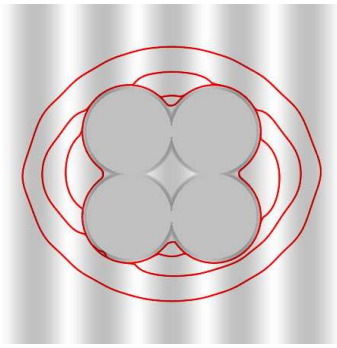
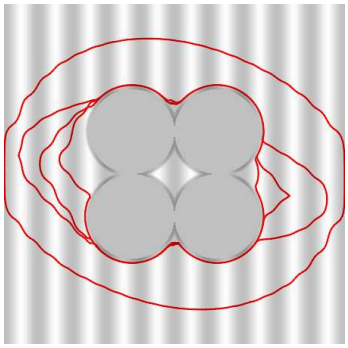
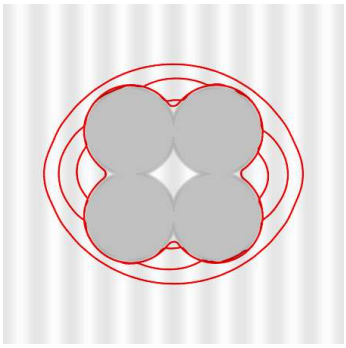
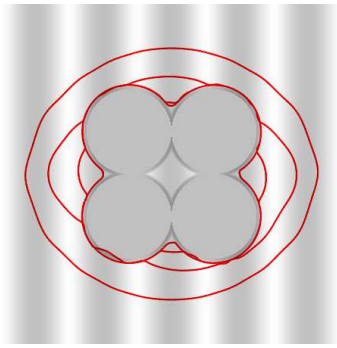
$N = 2,\ A = 1.0$	$N = 2,\ A = 0.3$	$N = 4,\ A = 1.0$
		
$k = 100, 350, 550, 2500$ $J_0 = 3.800$ $J = 0.411$	$k = 35, 50, 65, 101$ $J_0 = 5.852$ $J = 0.355$	$k = 35, 50, 70, 104$ $J_0 = 5.237$ $J = 0.346$
		
$k = 50, 200, 550, 2900$ $J_0 = 3.800$ $J = 0.646$	$k = 35, 45, 55, 97$ $J_0 = 5.852$ $J = 0.341$	$k = 30, 50, 65, 90$ $J_0 = 5.237$ $J = 0.348$

Figure 5.12: The segmentation results for images with oscillating patterns in the background. The pattern is given by $A\cos\left(\frac{\pi x_1}{N\varepsilon_{edge}}\right)$. The top and bottom rows show the results for the L^2 and H^1 flows respectively. For each experiment we display the curves for four of the iterations at different stages of the optimization. We observe that, for $N = 2$ and $A = 1$, the variation of oscillation is comparable to the variation at the edges. This causes the curves to get stuck along the background patterns. However for the choices of $N = 2$, $A = 0.3$ and $N = 4$, $A = 1$, the segmentation results are successful.

patterns. Otherwise the method may perform poorly.

5.1.2 Experiments with Real Images

In this section we present experiments demonstrating the effectiveness of the method on real images. We observed in §5.1.1.4 that some preprocessing is necessary to increase the effectiveness of the method. For this we apply two smoothing filters to the image consecutively. These are selective Gaussian blur and Gaussian blur. The idea of selective Gaussian blur is to consider pixel neighborhoods of radius ρ_{SG} and to apply a Gaussian blur if the variation between the pixel and its surrounding pixels exceeds a given threshold δ_{SG}. We use this to destroy weaker variations in the image while maintaining the sharp edges. Then we apply a Gaussian blur of radius ρ_G to smooth out the edges in order to improve the performance. We compute the image derivatives by applying the Scharr derivative filter. We evaluate inter-pixel values of image intensity using linear interpolation. We experiment on four images: the large bacteria, the small bacteria, the tiger and the vertebra images. The first two are from Wikimedia Commons (http://commons.wikimedia.org), the third is from Berkeley Segmentation Data Set (http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds), the last image is from [32]. The parameters used for each experiment are as follows:

image	ρ_{SG}	δ_{SG}	ρ_G	γ_0	γ	λ	L^2 *tol*	H^1 *tol*
large bacteria	30	20	10	0.005	1	1.0	10^{-2}	10^{-5}
small bacteria	30	20	5	0.01	1	1.0	10^{-3}	10^{-6}
tiger	30	50	10	0.01	1	1.0	10^{-2}	10^{-4}
vertebra	20	50	5	0.003	-1	0.5	10^{-4}	10^{-4}

The original images, the preprocessed images and the corresponding edge indicator functions H are given in Figure 5.13. Now we describe each experiment and the results obtained. I

Example 1: Large Bacteria. In this experiment we try to capture the boundary of a single strip of bacteria. Both L^2 and H^1 flows terminate successfully. We observe that H^1 flow yields a smoother evolution and terminates in fewer iterations. Moreover it captures the concavity better and results in a smaller final energy (see Figures 5.14 and 5.15).

Example 2: Small Bacteria. This is the image of multiple disconnected bacteria. To detect these the curve has to split into several separate curves. In this way both L^2 and H^1 flows manage to capture the boundaries of all the bacteria inside the initial curve. This is shown in Figures 5.16 and 5.17.

Example 3: Tiger. This is the image of a natural scene with a tiger. We want to segment the tiger including the thin tail and the concavities. Both L^2 and H^1 flows perform well, but the numbers of iterations for the two flows are comparable in contrast to previous experiments (see Figures 5.18 and 5.19).

Example 4: Vertebra. The last image we consider is from medical domain: the vertebra image. The typical practice in medical imaging is to start with a number seeds or initial curves that are inside the objects to be segmented. These expand and capture the boundary of the objects. We follow this and present two sets of experiments: one with a single seed and another with three seeds. We capture the boundary completely in both cases. But the experiments with three seeds terminate more quickly as all the seeds work locally at the beginning and then merge to capture the complete boundary.

In all the experiments we observe that H^1 flow converges faster (but not dramatically faster) than L^2 flow. It also produces smoother evolutions.

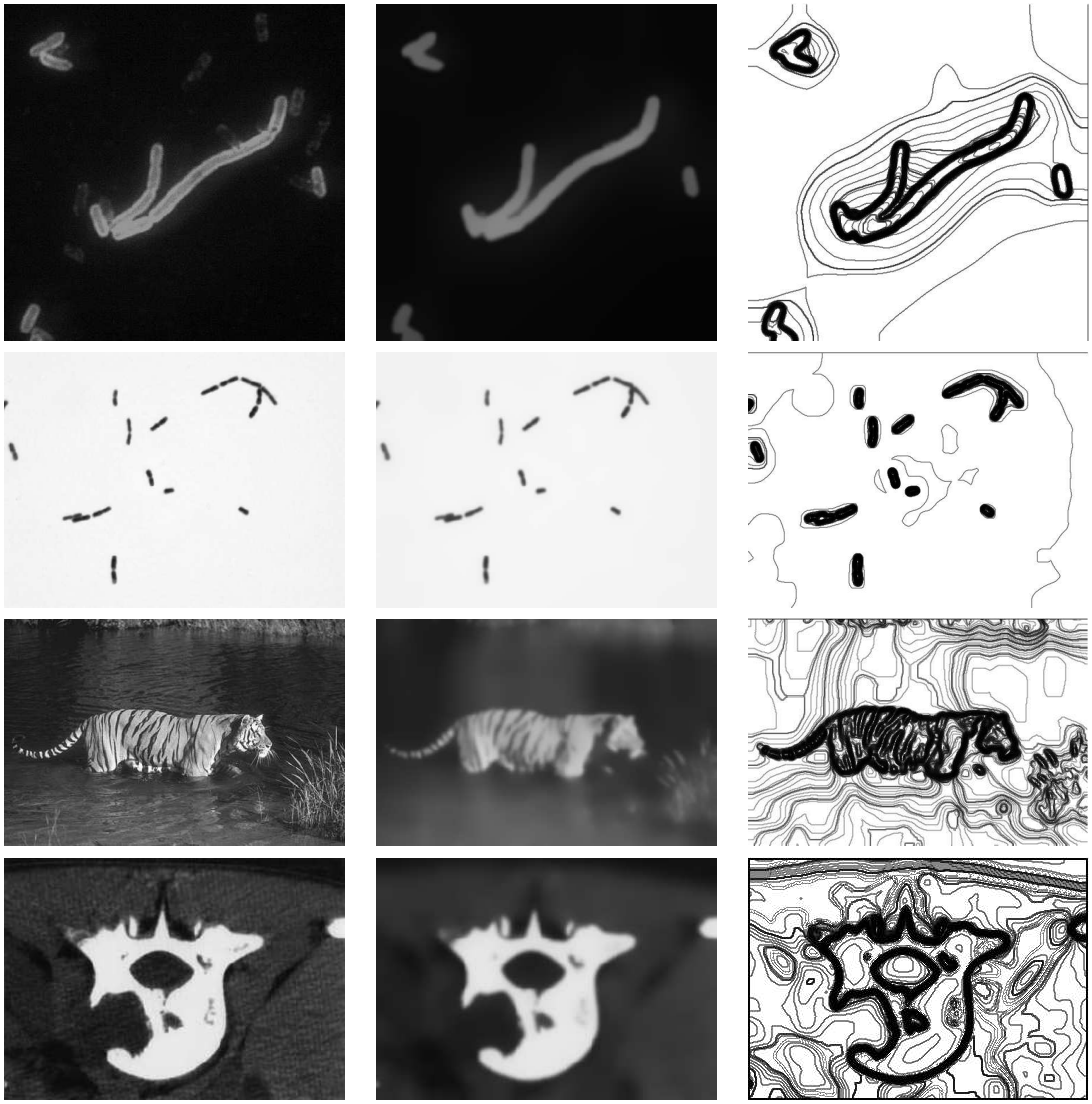
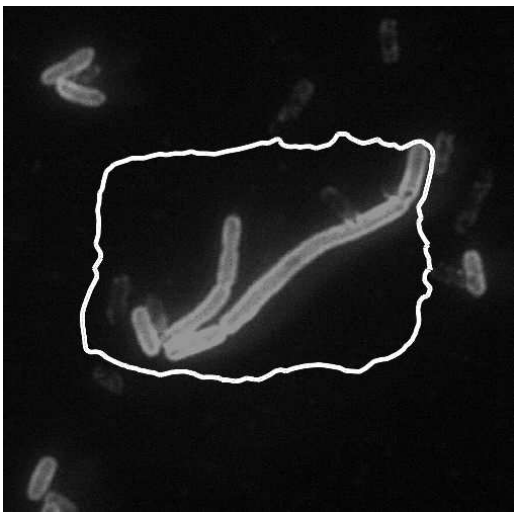
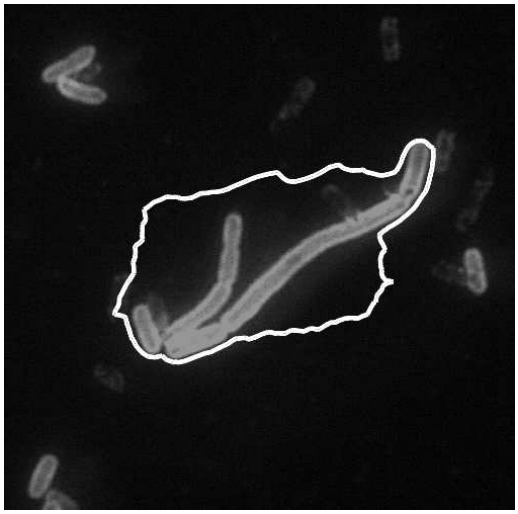


Figure 5.13: From top to bottom, the large bacteria image, the small bacteria image, the tiger image and the vertebra image. The left column shows the images before preprocessing. The middle column shows the images after processing. The right column shows the corresponding edge indicator functions.

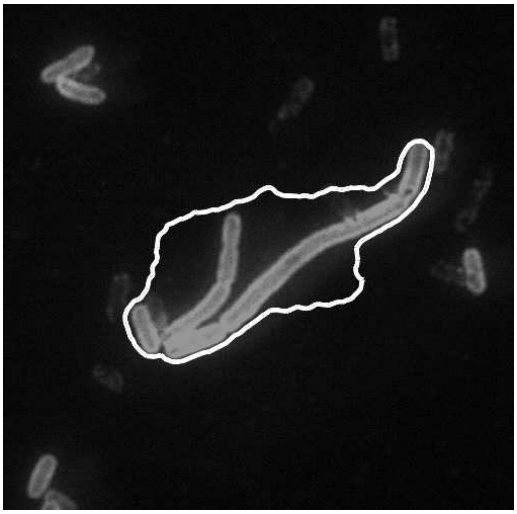
$k = 1,\ J = 1.794$



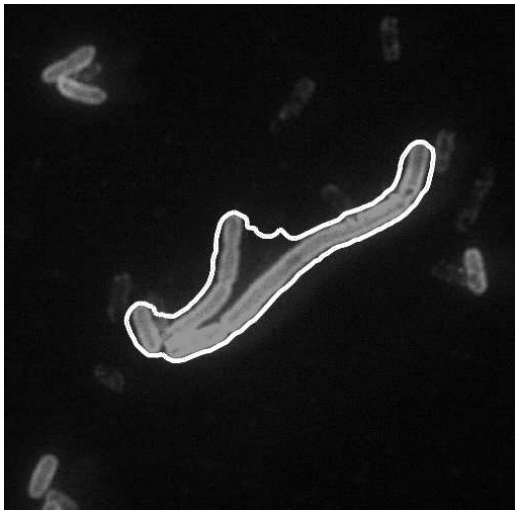
$k = 131,\ J = 1.148$



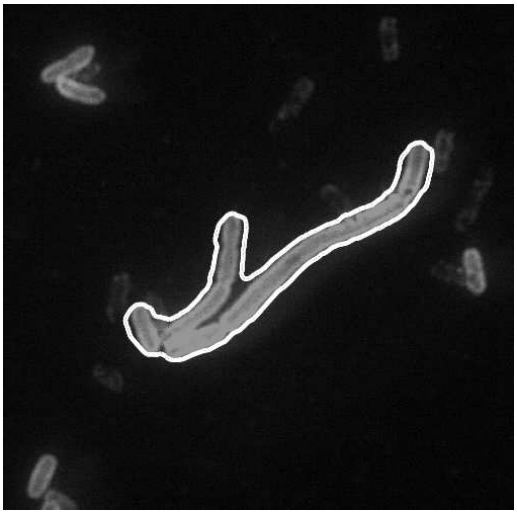
$k = 321,\ J = 0.695$



$k = 421,\ J = 0.490$



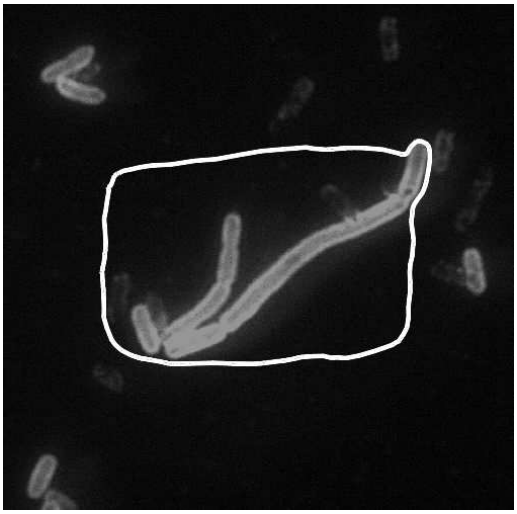
$k = 631,\ J = 0.260$



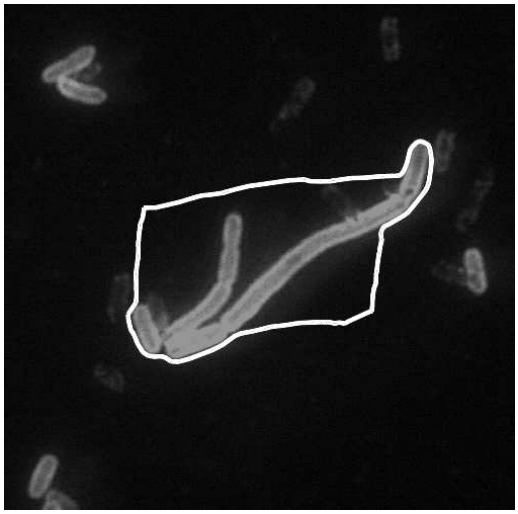
$k = 865,\ J = 0.224$

Figure 5.14: The evolution of the curve given by L^2 flow for the large bacteria image.

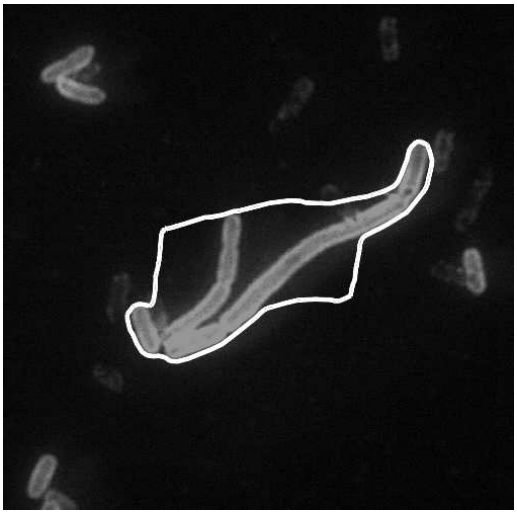
$k = 1,\ J = 1.777$



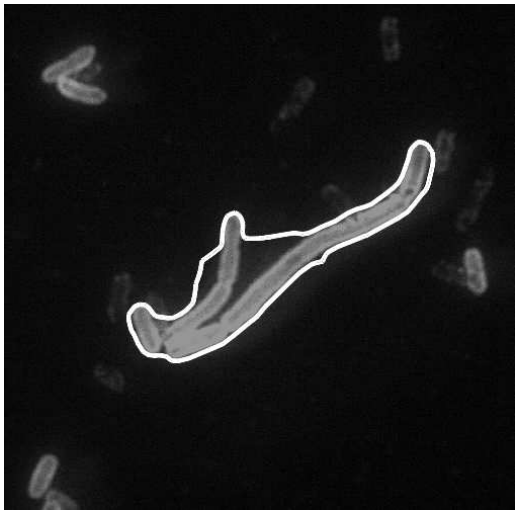
$k = 41,\ J = 0.976$



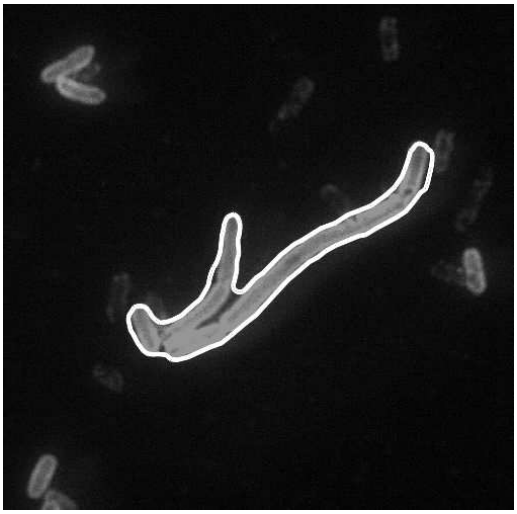
$k = 71,\ J = 0.596$



$k = 91,\ J = 0.425$



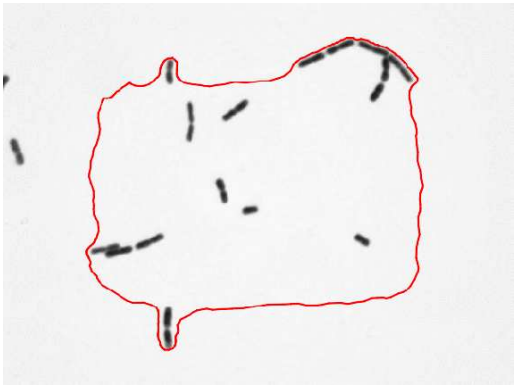
$k = 131,\ J = 0.252$



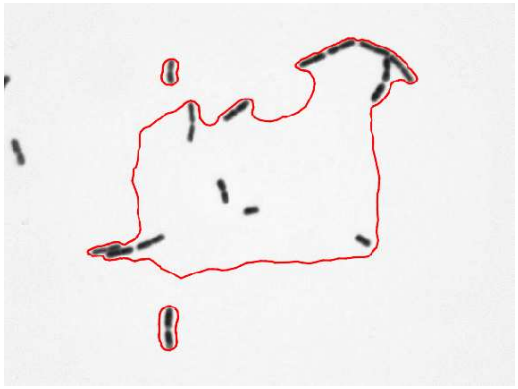
$k = 276,\ J = 0.194$

Figure 5.15: The evolution of the curve given by H^1 flow for the large bacteria image.

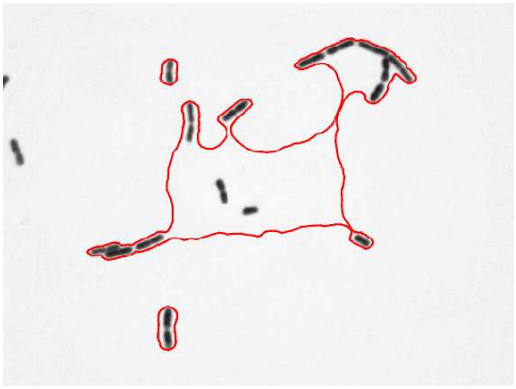
$k = 1,\ J = 4.263$



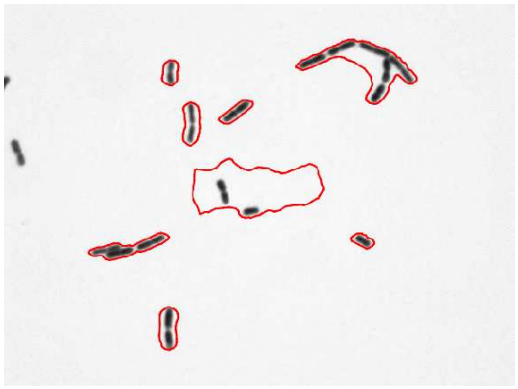
$k = 81,\ J = 2.132$



$k = 131,\ J = 1.232$



$k = 171,\ J = 0.659$

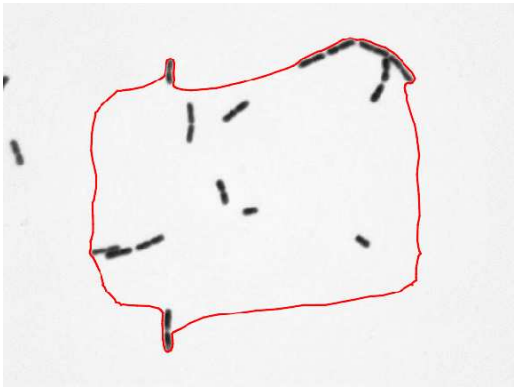


$k = 201,\ J = 0.324$

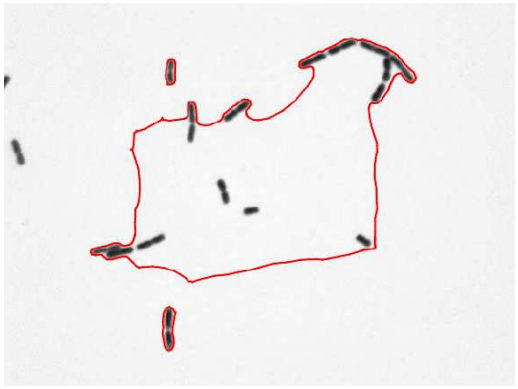
$k = 453,\ J = 0.160$

Figure 5.16: The evolution of the curve given by L^2 flow for the small bacteria image.

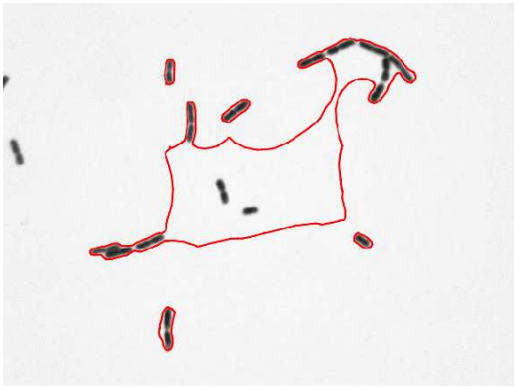
$k = 1,\ J = 4.263$



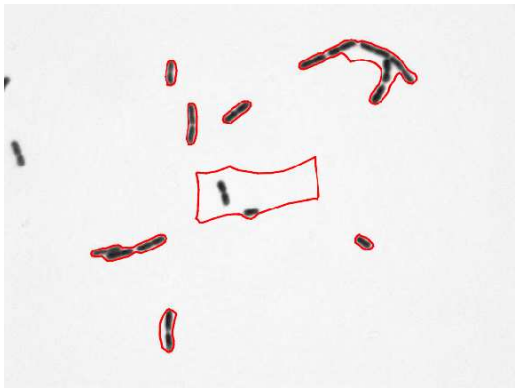
$k = 81,\ J = 2.092$



$k = 131,\ J = 1.198$



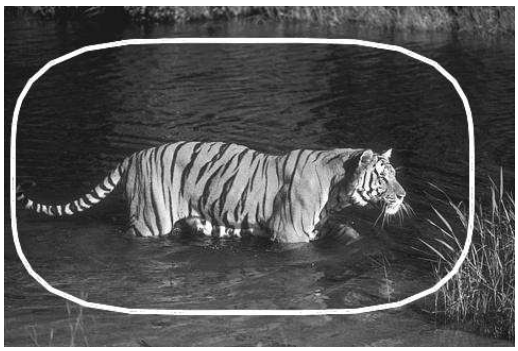
$k = 171,\ J = 0.622$



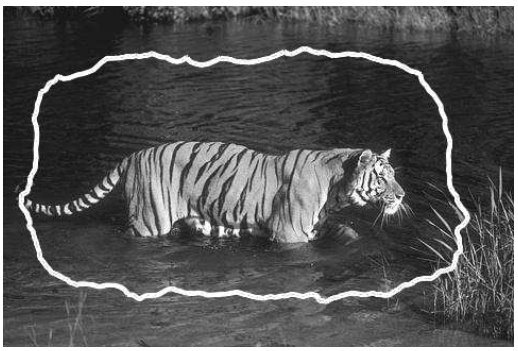
$k = 201,\ J = 0.265$

$k = 315,\ J = 0.125$

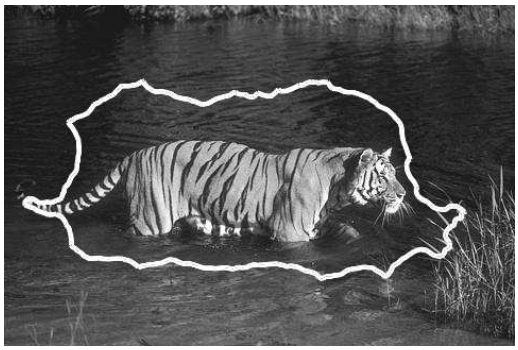
Figure 5.17: The evolution of the curve given by H^1 flow for the small bacteria image.



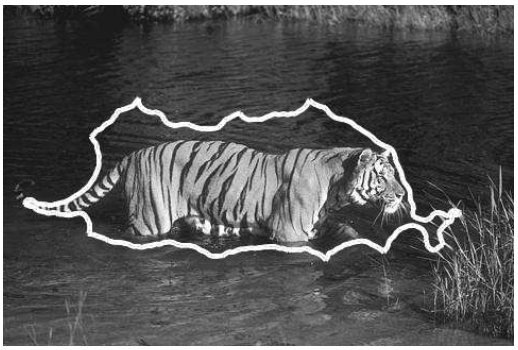
$k = 1,\ J = 4.012$



$k = 31,\ J = 3.299$



$k = 91,\ J = 2.097$

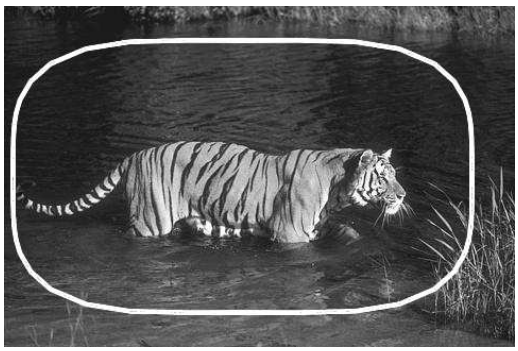


$k = 131,\ J = 1.437$

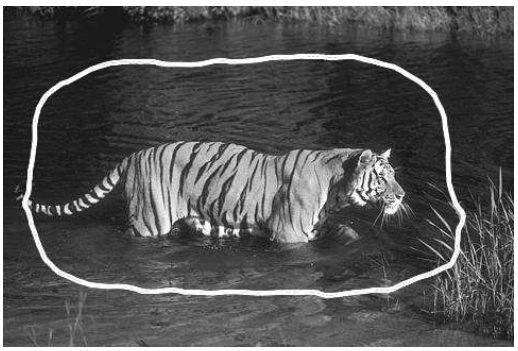
$k = 171,\ J = 0.996$

$k = 231,\ J = 0.819$

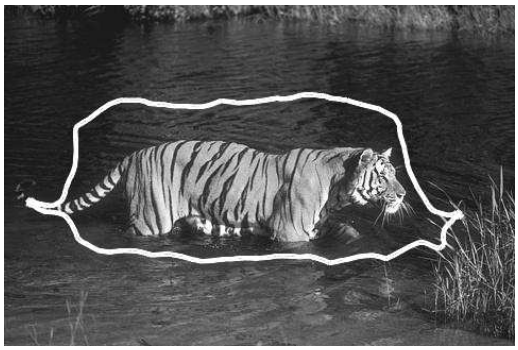
Figure 5.18: The evolution of the curve given by L^2 flow for the tiger image.



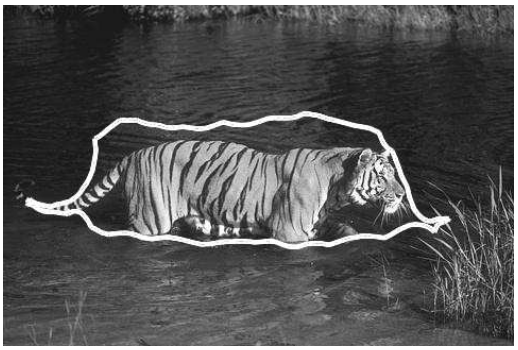
$k = 1,\ J = 4.012$



$k = 31,\ J = 3.199$



$k = 91,\ J = 1.848$

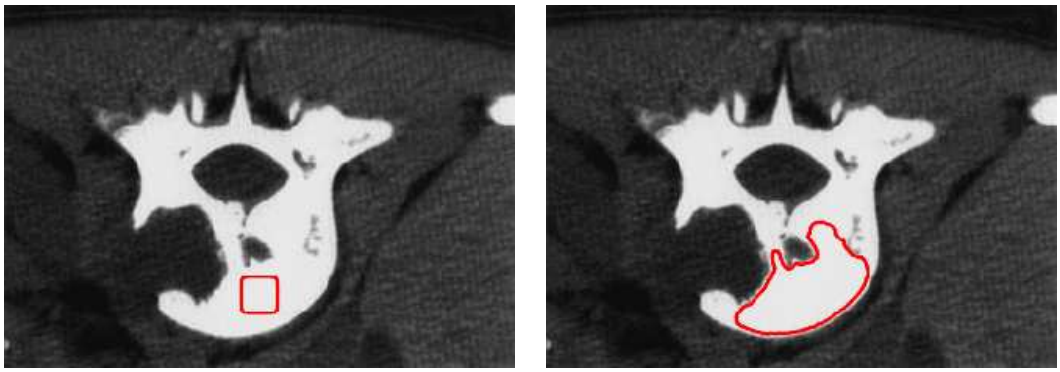


$k = 121,\ J = 1.289$

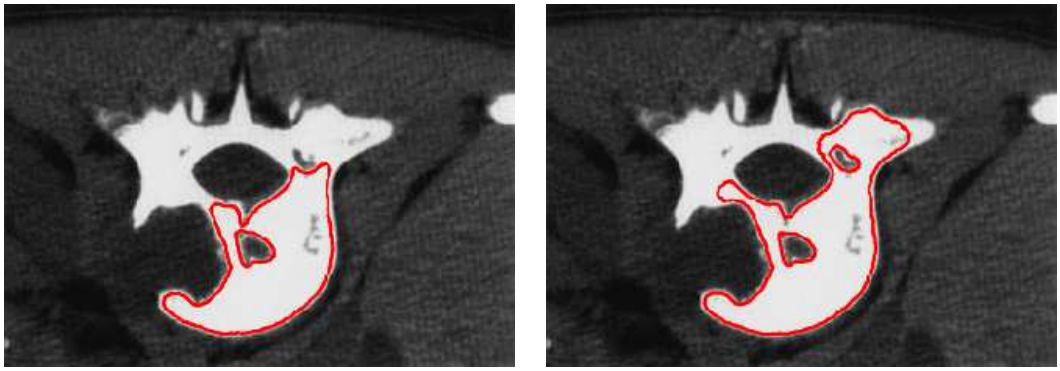
$k = 151,\ J = 0.956$

$k = 202,\ J = 0.803$

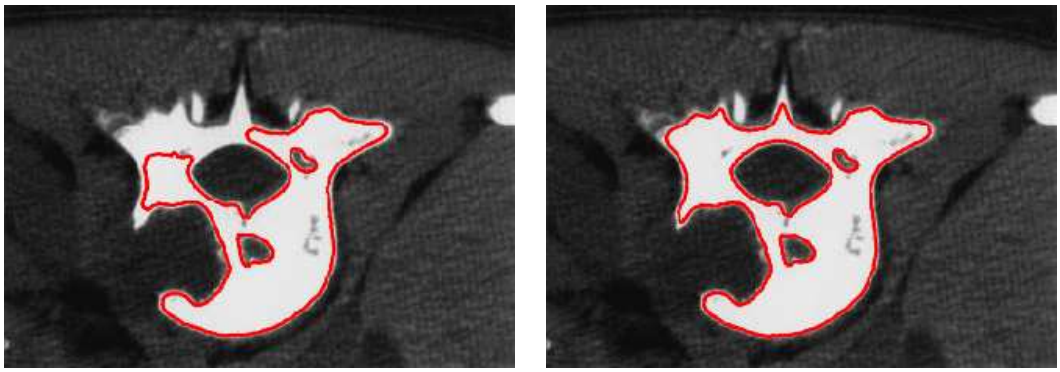
Figure 5.19: The evolution of the curve given by H^1 flow for the tiger image.



$k = 1,\ J = -0.040$ $k = 111,\ J = -0.237$

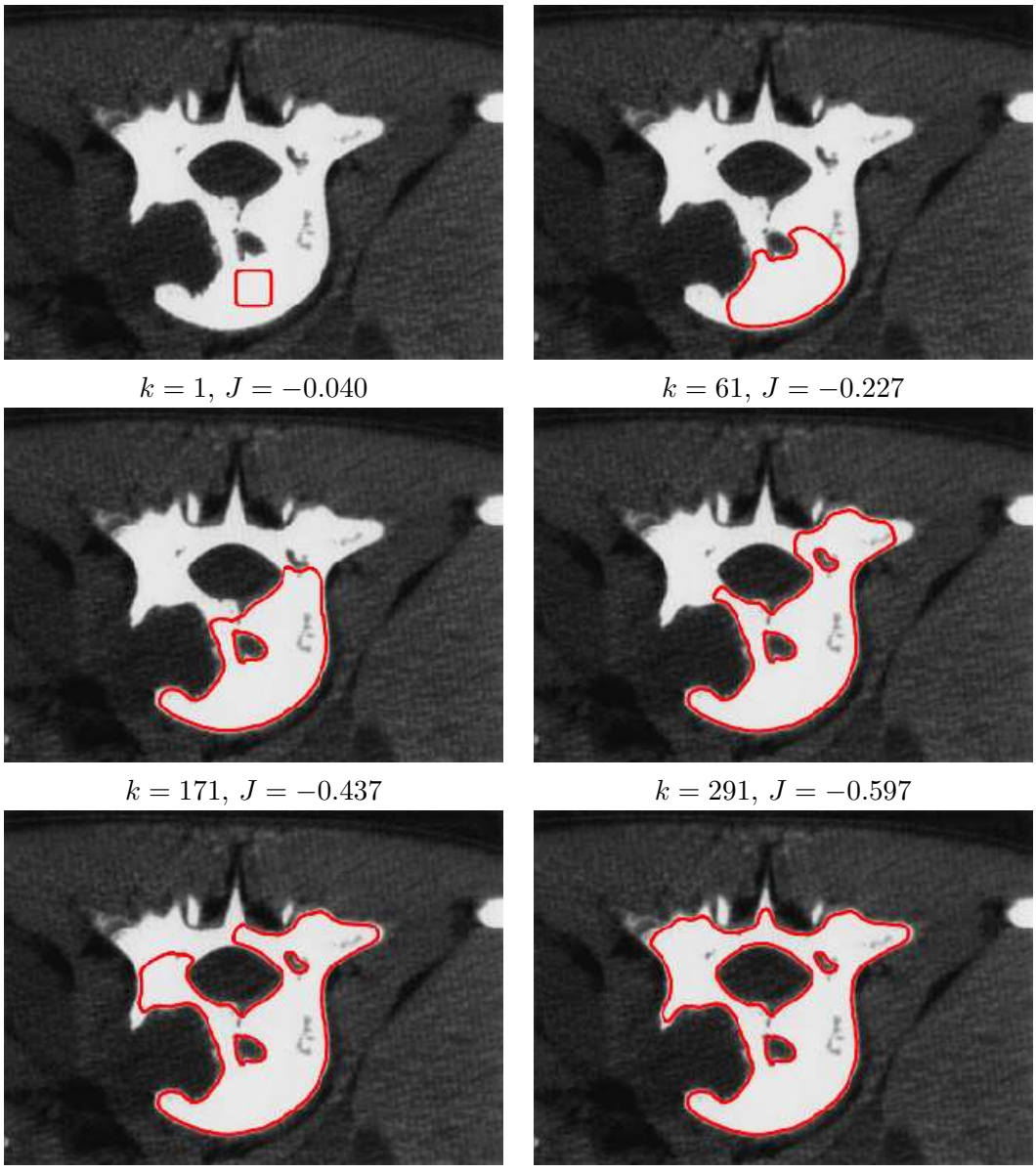


$k = 241,\ J = -0.430$ $k = 361,\ J = -0.564$



$k = 431,\ J = -0.667$ $k = 581,\ J = -0.811$

Figure 5.20: The evolution of the curve given by L^2 flow for the vertebra image starting with a single seed.

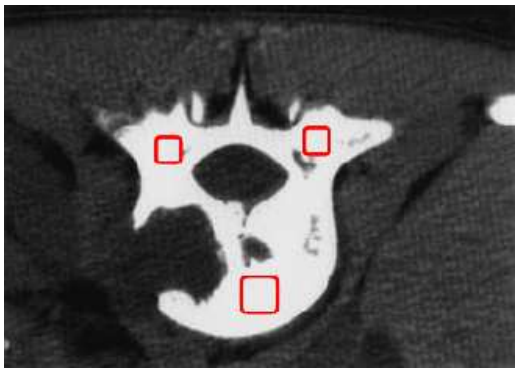


$k = 1,\ J = -0.040$ $k = 61,\ J = -0.227$

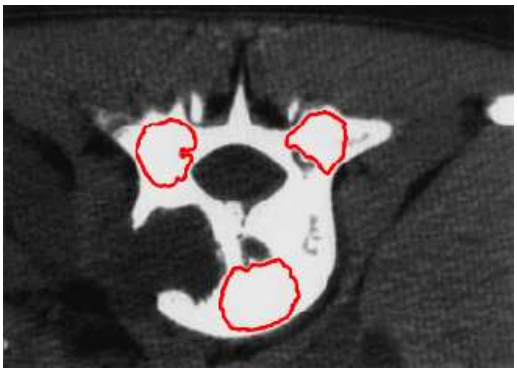
$k = 171,\ J = -0.437$ $k = 291,\ J = -0.597$

$k = 371,\ J = -0.711$ $k = 441,\ J = -0.832$

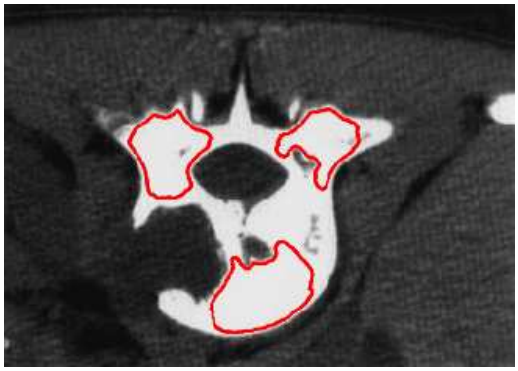
Figure 5.21: The evolution of the curve given by H^1 flow for the vertebra image starting with a single seed.



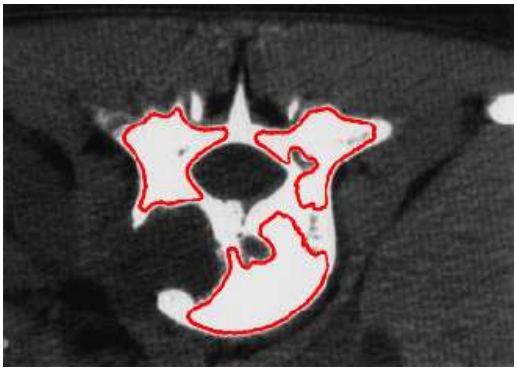
$k = 1,\ J = -0.077$



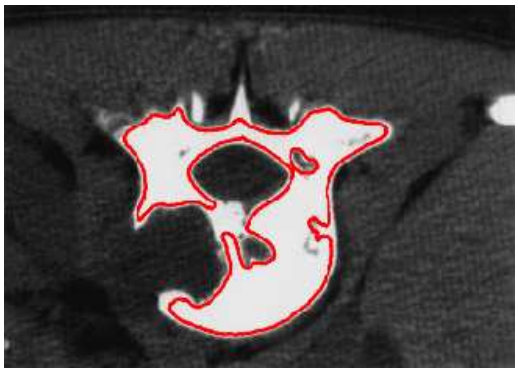
$k = 61,\ J = -0.281$



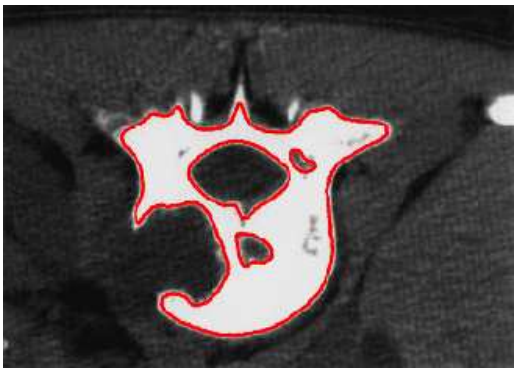
$k = 111,\ J = -0.405$



$k = 191,\ J = -0.568$

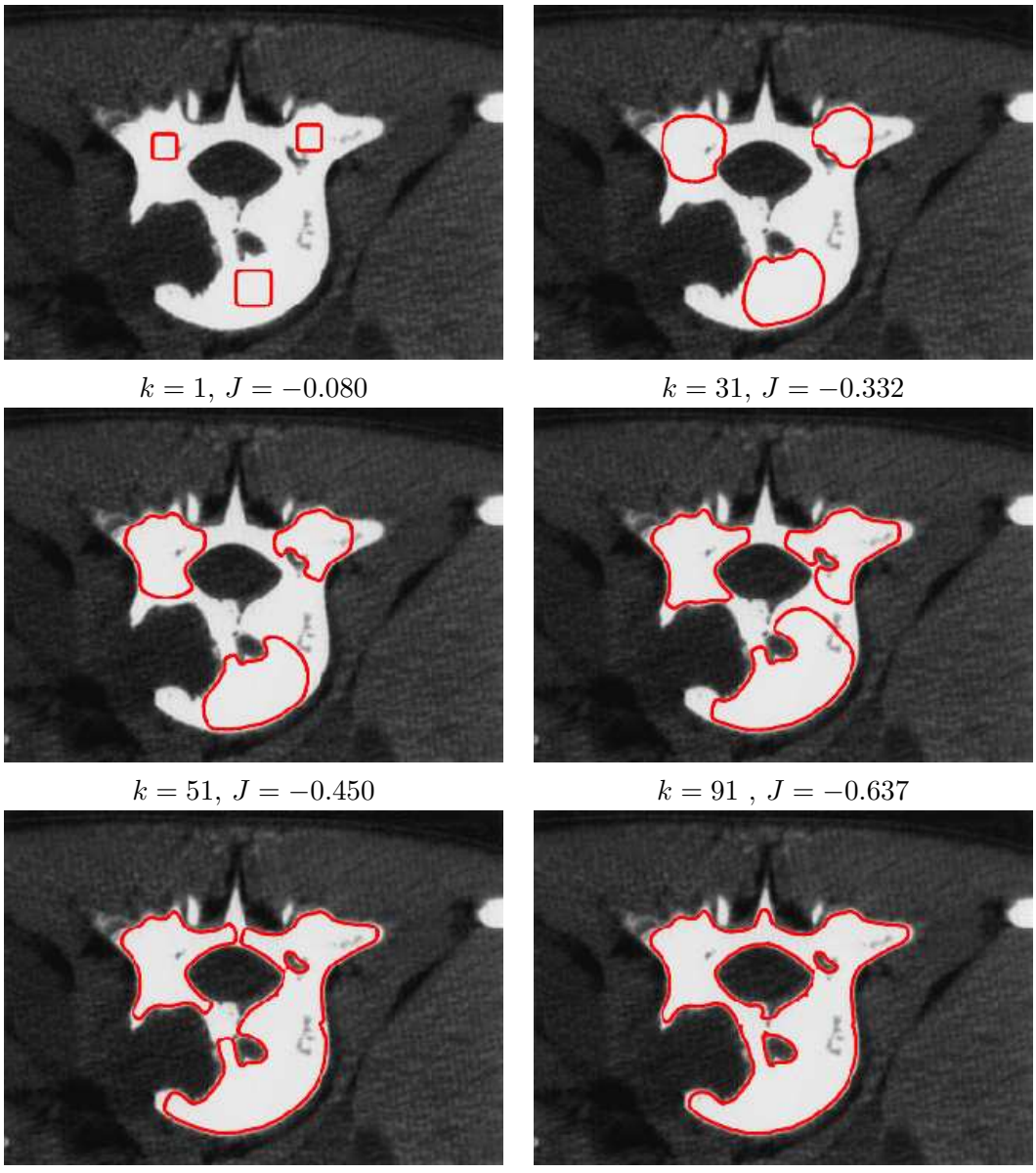


$k = 271,\ J = -0.705$



$k = 451,\ J = -0.803$

Figure 5.22: The evolution of the curve given by L^2 flow for the vertebra image starting with three seeds.



$k = 1,\ J = -0.080$ $k = 31,\ J = -0.332$

$k = 51,\ J = -0.450$ $k = 91\ ,\ J = -0.637$

$k = 131,\ J = -0.760$ $k = 191,\ J = -0.818$

Figure 5.23: The evolution of the curve given by H^1 flow for the vertebra image starting with three seeds.

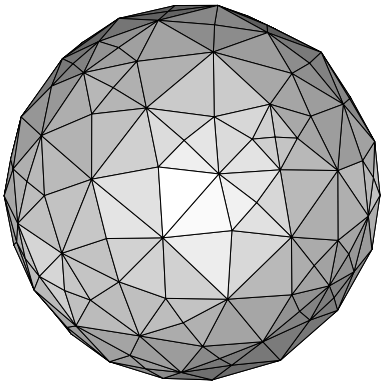
5.1.3 Experiments with 3d Images

In this section we consider the model (5.1) for 3d volume images. This is of importance particularly for medical imaging, but it is becoming more and more relevant in other branches of science too as 3d data that need to be visualized continue to be produced by various experiments and simulations.

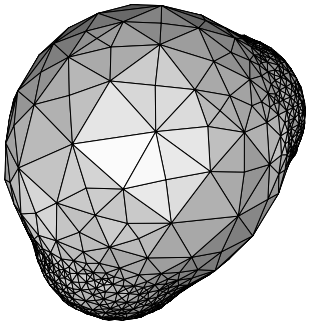
We demonstrate our method on two synthetic experiments. For these we use the H^1 flow. Our preliminary 3d experiments follow our findings in the 2d experiments. H^1 flow produces smoother evolutions compared with L^2 flow and converge in fewer iterations. This makes the H^1 flow the method of choice for the more costly 3d experiments.

In the first experiment, we try to capture the boundary of two identical balls touching each other. A distinct feature of this image is that there is a cusp around the touching point. We start the evolution with a coarse surface and the surface refines as it starts capturing the object. In the end the boundary of the object is acquired very well including the region around the cusp. We present this result in Figure 5.24.

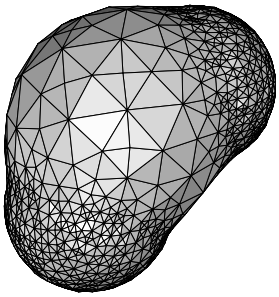
In the second experiment, we want to capture a rectangular prism. We start with the unit sphere. This result is given in Figure 5.25. We see again in this example how space adaptivity refines the mesh as the surface captures the mesh. In particular we see that the method adds more resolution at the sharp features, such as the edges and the corners. But it coarsens the mesh at the faces where a smaller number of elements is enough to represent the object accurately.



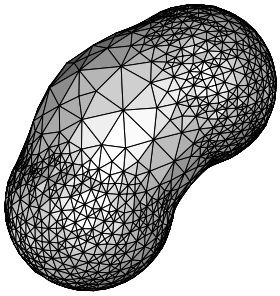
$k = 12$
$J = 1.802$



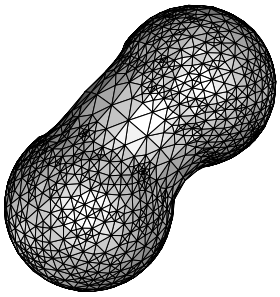
$k = 20$
$J = 0.926$



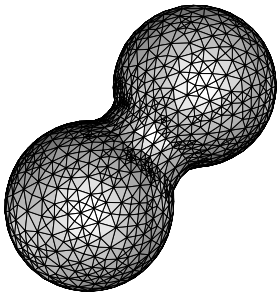
$k = 25$
$J = 0.590$



$k = 30$
$J = 0.407$

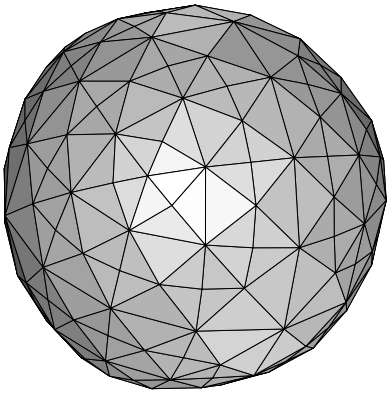


$k = 35$
$J = 0.317$

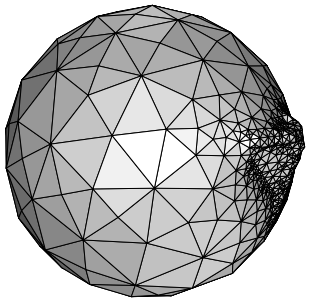


$k = 59$
$J = 0.278$

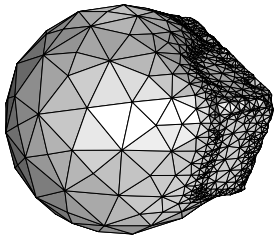
Figure 5.24: Detection of a 3D object consisting of two touching balls with weighted H^1 flow. Note the effect of space adaptivity; we start with a relatively coarse spherical surface and the mesh refines as it detects the object boundary.



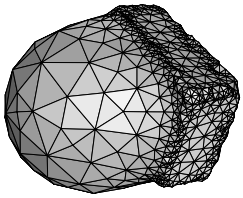
$k = 5$
$J = 10.831$



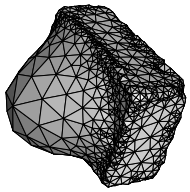
$k = 50$
$J = 4.441$



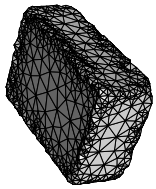
$k = 85$
$J = 2.780$



$k = 125$
$J = 1.517$



$k = 160$
$J = 0.500$



$k = 241$
$J = 0.024$

Figure 5.25: Detection of a prism in 3d with weighted H^1 flow. We can see the effect of space adaptivity: the mesh is finer at edges and corner, but coarser at the faces.

5.2 The Mumford-Shah Model

In this section we use the Mumford-Shah functional in the form

$$J(\Gamma) = \sum_{i=1}^{2} \frac{1}{2} \int_{\Omega_i} \left((u_i - I)^2 + \mu |\nabla u_i|^2 \right) dx + \gamma \int_{\Gamma} dS. \tag{5.4}$$

This leads to the pair of boundary value problems (essentially the optimality conditions with respect to u_i)

$$\begin{cases} -\mu \Delta u_i + u_i &= I \quad \text{in} \quad \Omega_i \\ \partial_{\nu_i} u_i &= 0 \quad \text{on} \quad \partial\Omega_i \end{cases}$$

with $i = 1, 2$ to segment a given image into foreground objects and a background region represented by Ω_1 and Ω_2 respectively. This separation of foreground and background will also yield a piecewise smooth approximation to the image given by

$$u = u_1 \chi_{\Omega_1} + u_2 \chi_{\Omega_2}.$$

Recall that the first shape derivative of (5.4) is given by

$$dJ(\Gamma; V) = \int_{\Gamma} \Big(\frac{1}{2} \left[\!\left[|u - I|^2 \right]\!\right] + \frac{\mu}{2} \left[\!\left[|\nabla_\Gamma u|^2 \right]\!\right] + \gamma\kappa \Big) V dS.$$

The second shape derivative is

$$\begin{aligned} d^2 J(\Gamma; V, W) &= \gamma \int_{\Gamma} \nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \frac{\mu}{2} \int_{\Gamma} \left(\kappa \left[\!\left[|\nabla u|^2 \right]\!\right] + \partial_\nu \left[\!\left[|\nabla u|^2 \right]\!\right] \right) V W dS \\ &+ \frac{1}{2} \int_{\Gamma} \left(\kappa \left[\!\left[|u - I|^2 \right]\!\right] + \partial_\nu \left[\!\left[|u - I|^2 \right]\!\right] \right) V W dS \\ &+ \int_{\Gamma} \left(\left[\!\left[(u - I) u'_W \right]\!\right] + \mu \left[\!\left[\nabla u \cdot \nabla u'_W \right]\!\right] \right) V dS. \end{aligned}$$

We use this to introduce a weighted H^1 scalar product

$$\begin{aligned}\langle V, W\rangle_{H^1} &= \gamma \int_\Gamma \nabla_\Gamma V \cdot \nabla_\Gamma W dS \\ &+ \frac{1}{2}\int_\Gamma \left(\kappa(\left[\!\left[|u-I|^2\right]\!\right] + \mu \left[\!\left[|\nabla u|^2\right]\!\right]) + \partial_\nu \left[\!\left[|u-I|^2\right]\!\right]\right)_+ VW dS\end{aligned}$$

where $(\cdot) = \max(\cdot, \epsilon)$ and we choose $\epsilon = 0.01$. Similar to the case with the H^1 flow for the minimal surface model, the main motivation for this choice is to ensure well-conditioning of $A_\alpha + M_\beta$ and such small values for ϵ are acceptable as γ, which weights A_α, is even smaller. Now since we have positive definiteness of the scalar product, by setting $b(V, W) = \langle V, W\rangle_{H^1}$, we can solve the system (3.4) and obtain an inexact Newton's descent direction. This will be our choice in the optimization as our experiments will demonstrate the superiority of the weighted H^1 flow to L^2 flow.

Let us mention some of the practical issues related to implementation. The first is the time step selection procedure, which we have chosen to be backtracking described in Algorithm 1. We set $\tau_{min} = 0.05, 0.005$ and $\tau_{max} = 1.0, 0.1$ for L^2 and H^1 flows respectively. As stopping criterion, we use the shape derivative criterion (5.2) with $tol_{rel} = 0$ and suitable tol values.

As we choose a Lagrangian approach to move the boundary, we need a way to track the domain. We choose to generate a triangulation of the domain at each time step using the 2d mesh generation program TRIANGLE by Shewchuk [62]. In general this tends to create meshes that are very coarse away from the surface and this brings the possible problems of accuracy. To compensate for this we employ the data-driven space adaptivity procedure described in Algorithm 8 to refine the

domain mesh when necessary. For the curve we use the geometry-driven space adaptivity procedure described in Algorithm 6. The parameters for these are $\epsilon_{data} = 5 \times 10^{-2}$, $\epsilon_{geom} = 0.02$, $\gamma_R^G = 0.7$, $\gamma_C^G = 0.3$. The time step control routine is used as well with $\epsilon_\tau = 0.8$.

Furthermore we have found a *multilevel approach* to be very beneficial for this optimization problem. The main idea for this is to keep the computations as coarse as possible when we are far away from the optimum and to refine the computations as we get closer. The motivation is twofold. First from the time step side: very careful time steps at the beginning of the optimization is not worth the effort. This is in particular true with backtracking as we may need to solve the domain PDE several times to compute the energy as part of backtracking. So we prefer to use N_{coarse} number of fixed time steps at the beginning. This hopefully brings us closer to the optimum. In this stage, space adaptivity in domain is also switched off and space adaptivity on the curve is executed with $\epsilon_{coarse} = C_{coarse}\epsilon_{geom}, \quad C_{coarse} \geqslant 1$. So these initial iterations are cheap. Another very important benefit of this coarse stage is that it helps avoid local minima.

After the initial coarse stage, we take a continuation approach to tighten the tolerance parameters of the space adaptivity procedures. We enforce better resolution as we approach the minimum. However an important matter here is that we do not really have a concrete way of determining whether we are close to the minimum. Nevertheless we have a clue, which is the shape derivative. Since the shape derivative becomes small close to the minimum, we can use this as a means of determining when we should enforce higher resolution. For this we choose a sequence

of derivative thresholds $\infty = d_n > d_{n-1} > \ldots > d_1 > d_0 = 0$ and corresponding adaptivity tolerance factors $C_n \geqslant C_{n-1} \geqslant \ldots \geqslant C_1 \geqslant C_0 = 1$ and corresponding time steps $\tau_n \geqslant \tau_{n-1} \geqslant \ldots \geqslant \tau_1$. Then at a given iteration k, if the magnitude of the shape derivative $|dJ(\Gamma_k; V_k)|$ is the interval $[d_i, d_{i+1}]$, $i > 0$, the adaptivity tolerance is set to be C_i times the original tolerance and the time step is set to be τ_i. If $|dJ(\Gamma_k; V_k)|$ is in $[d_1, d_0]$, we use backtracking for time step selection. The multilevel method obtained this way for (5.4) is described in Algorithm 17. In our experiments we found that the use of two levels gives satisfactory results.

Algorithm 17 : Solver for Mumford-Shah

switch off adaptivity in domain
set $\epsilon_{coarse} = C_{coarse}\epsilon_{geom}$ for the curve, $C_{coarse} \geqslant 1$
for iter = 1 to N_{coarse} **do**
  adapt the curve Γ with ϵ_{coarse}
  solve PDE in Ω_1 and Ω_2
  solve for velocity V on Γ
  move curve Γ with V and τ_{coarse}
end for
switch on adaptivity in domain
compute $dJ(\Omega; V)$
repeat
  if $d_i \leqslant |dJ(\Gamma; V)| < d_{i+1}, \quad i = 0, \ldots, n-1$ **then**
    adapt the curve Γ with relaxed tolerance $C_i\epsilon_{geom}$
  end if
  solve PDE in Ω_1 and Ω_2
  solve for velocity V on Γ
  if $d_i \leqslant |dJ(\Omega; V)| < d_{i+1}, \quad i = 1, \ldots, n-1$ **then**
    set $\tau = \tau_i$
  else { $0 \leqslant |dJ(\Gamma; V)| < d_1$ }
    use backtracking to choose τ
  end if
  move curve Γ with V and τ
until $|dJ(\Gamma; V)| < tol$

5.2.1 L^2 flow vs H^1 flow

In our first set of experiments, we compare the convergence properties of the L^2 flow and the H^1 flow. In §5.1 we observed that the H^1 flow for model (5.1) yields superior results compared to the L^2 flow, in terms of number of iterations and smoothness of the evolution. We make the same observation with the Mumford-Shah model as well.

As a test image, we use the electron microscopy image of three pneumonia bacteria that we obtained from Wikimedia Commons (http://commons.wikimedia.org). We run the optimization for both L^2 and H^1 flows. The stopping tolerances are 10^{-5} and 10^{-3} respectively. The model parameters are $\mu = 5 \times 10^{-3}$, $\nu = 1.5 \times 10^{-3}$. The multilevel parameters are $N_{coarse} = 150$, $C_1 = C_2 = C_{coarse} = 1$, $d_1 = 5\,tol$, $d_2 = 20\,tol$. The time steps are $\tau_1 = 0.1$, $\tau_2 = 0.2$, $\tau_{coarse} = 1$ for L^2 flow, $\tau_1 = 0.01$, $\tau_2 = 0.02$, $\tau_{coarse} = 0.1$ for H^1 flow.

We present the image sequences showing the evolution of the curves for L^2 and H^1 flows in Figures 5.26 and 5.28 respectively. The evolution of the piecewise smooth approximations to the image are given in Figures 5.27 and 5.29. We see that H^1 flow terminates in 165 iterations, approximately one fourth the number of iterations for the L^2 flow. We find this to be a typical contrast between the two flows. In general the computation time for a single L^2 iteration is comparable to an H^1 iteration (as we can also deduce by examining the linear systems in Chapter 3). For this reason, we use the H^1 flow for the remaining experiments.

We also give the domain meshes used to solve the PDE in Figures 5.30 and

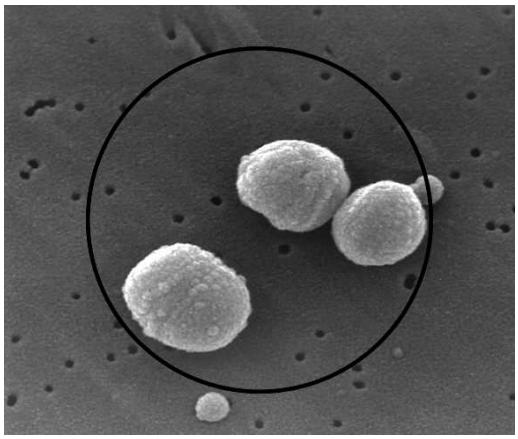
5.31. Figure 5.30 shows Ω_2, whereas Figure 5.31 shows Ω_1. In particular we can see how the mesh is affected after space adaptivity is switched on. In Figure 5.31, we see that the mesh is very coarse at the beginning. After the initial phase, as adaptivity is executed, the mesh becomes finer at the boundaries of the bacteria where the image varies more. It remains coarser elsewhere.

5.2.2 Denoising Properties

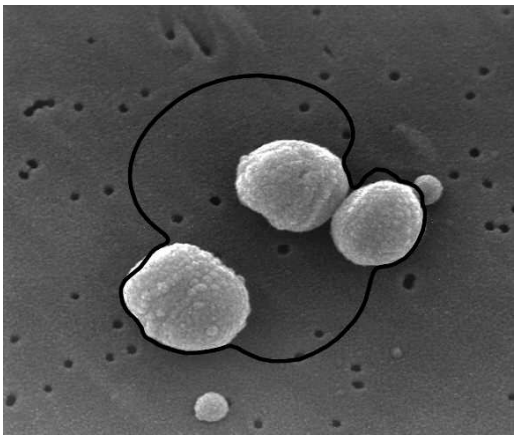
In this section we test the denoising properties of the Mumford-Shah model (5.4). We use the image of two jets flying, made available by Phillip Treweek at http://www.kiwiaircraftimages.com. From this we obtain a sequence of noisy images by applying white noise at levels of 0.1, 0.25 and 0.5 of image intensity. The original image and the sequence of noisy images are given in Figure 5.32.

The model parameters used for the experiments are: $\mu = 5 \times 10^{-3}$, $\nu = 1.5 \times 10^{-3}$; termination tolerance is $tol = 1 \times 10^{-3}$; multilevel parameters: $N_{coarse} = 150$, $C_1 = 1, C_2 = 2$, $C_{coarse} = 3$, $d_1 = 5\,tol$, $d_2 = 20\,tol$; time steps: $\tau_1 = 0.02$, $\tau_2 = 0.04$, $\tau_{coarse} = 0.1$.

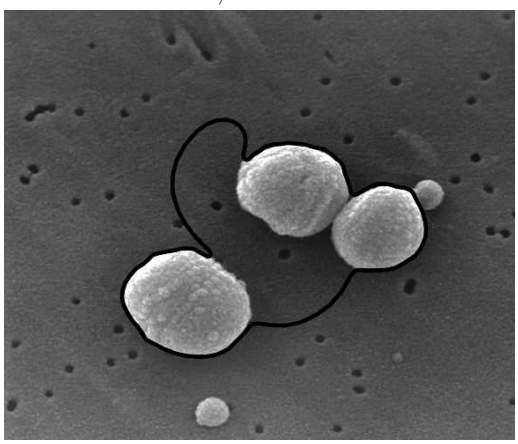
The result of the experiment for noise level 0.1 is given in Figures 5.33 and 5.34. As we run experiments for higher noise levels, we observe that the method takes more and more iterations to reach the optimum. In fact with second image at noise level 0.25, the experiment terminates prematurely using the termination tolerance of the previous experiment, $tol = 1 \times 10^{-3}$. The result is shown in Figure 5.35. We lower the tolerance to $tol = 2 \times 10^{-4}$ to obtain the expected result. The results



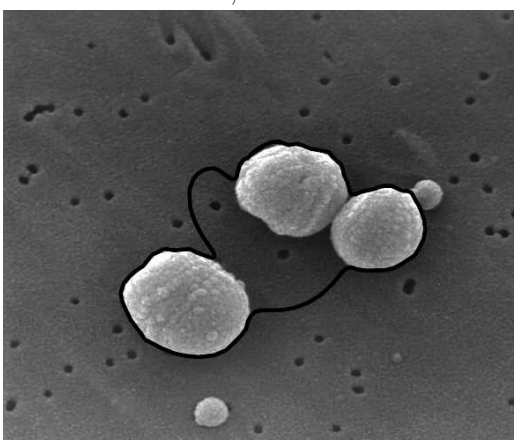
$k = 1,\ J = 0.0291$



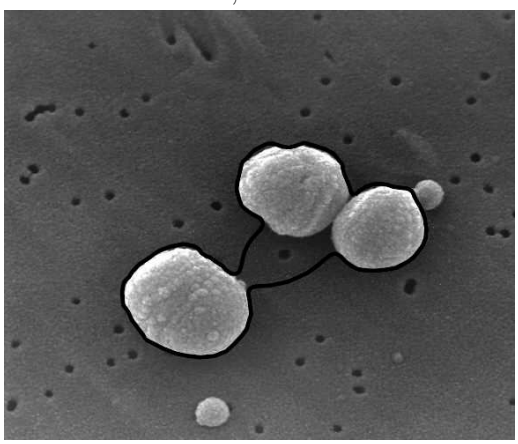
$k = 121,\ J = 0.0228$



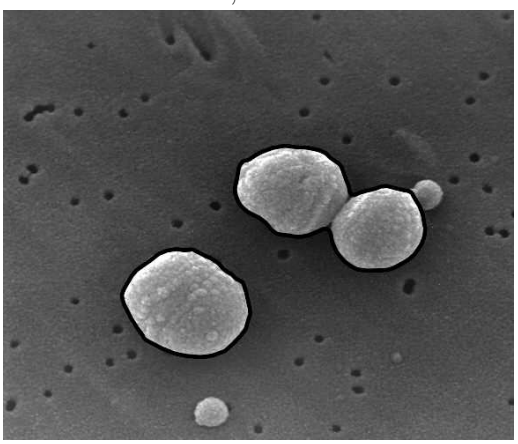
$k = 331,\ J = 0.0175$



$k = 441,\ J = 0.0162$

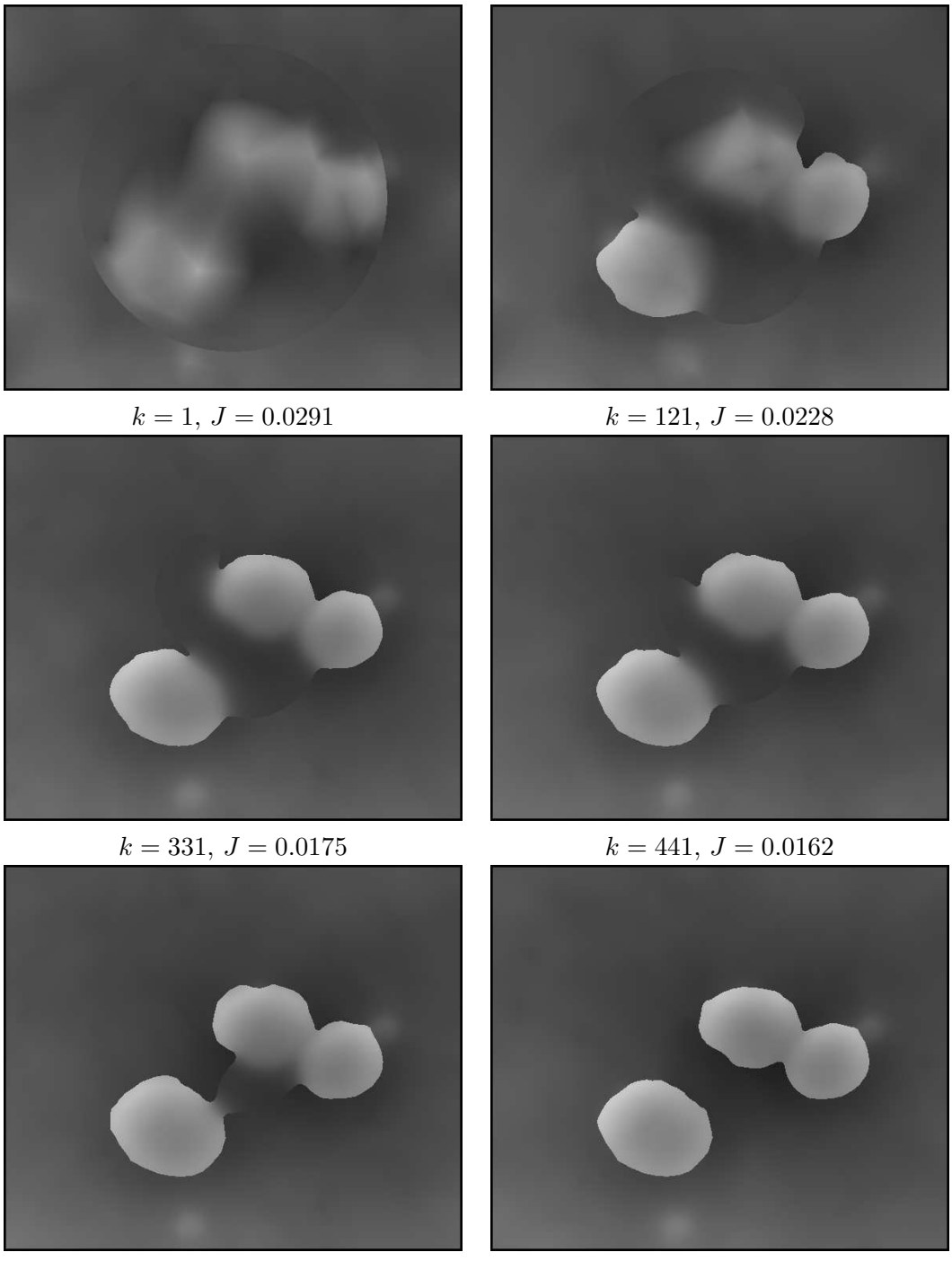


$k = 671,\ J = 0.0157$



$k = 841,\ J = 0.0130$

Figure 5.26: The evolution of the curve Γ given by L^2 flow for the bacteria image.

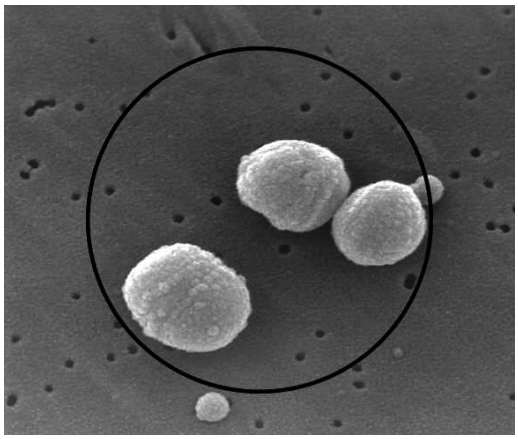


$k = 1,\ J = 0.0291$ $k = 121,\ J = 0.0228$

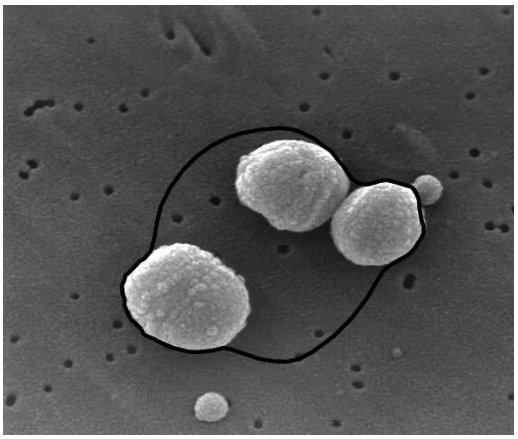
$k = 331,\ J = 0.0175$ $k = 441,\ J = 0.0162$

$k = 671,\ J = 0.0157$ $k = 841,\ J = 0.0130$

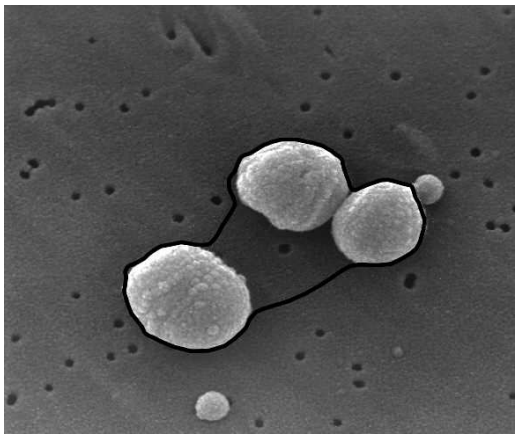
Figure 5.27: The evolution of the smooth approximation and domains Ω_1 and Ω_2 for the bacteria image given by L^2 flow.



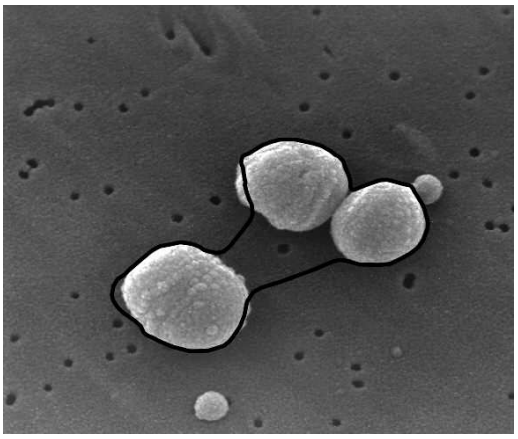
$k = 1,\ J = 0.0291$



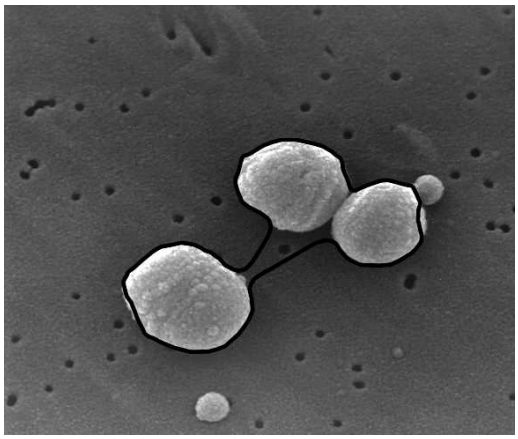
$k = 31,\ J = 0.0201$



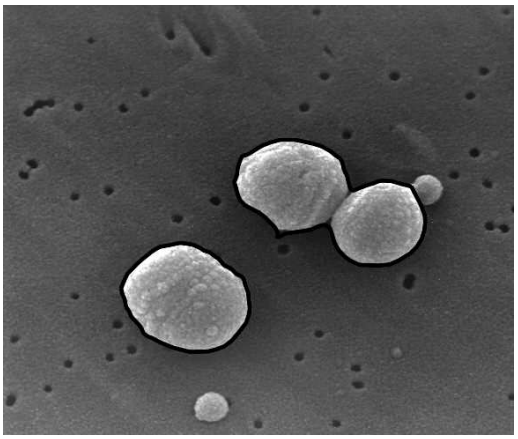
$k = 71,\ J = 0.0156$



$k = 91,\ J = 0.0160$

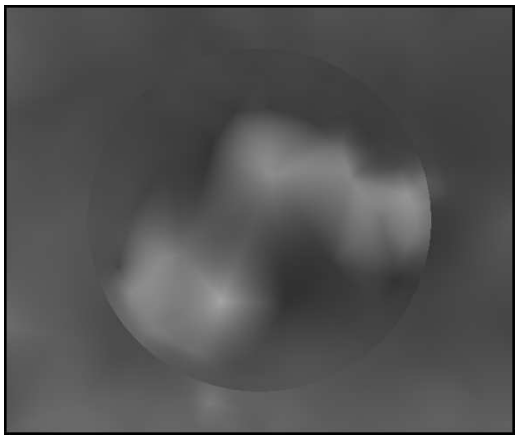


$k = 131,\ J = 0.0149$

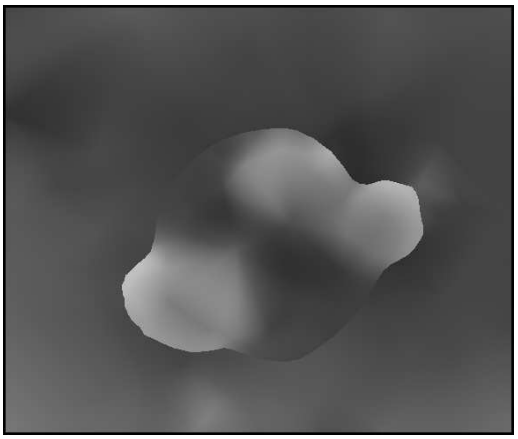


$k = 165,\ J = 0.0127$

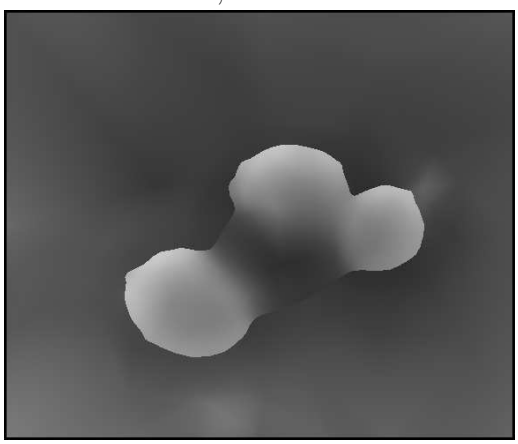
Figure 5.28: The evolution of the curve Γ given by H^1 flow for the bacteria image. The H^1 flow completes in much fewer iterations than the L^2 flow given in Figure 5.26.



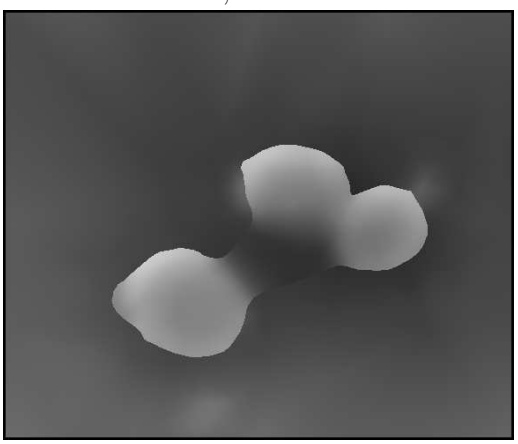
$k = 1,\ J = 0.0291$



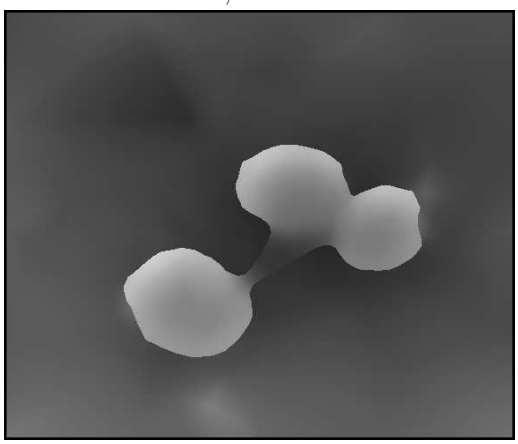
$k = 31,\ J = 0.0201$



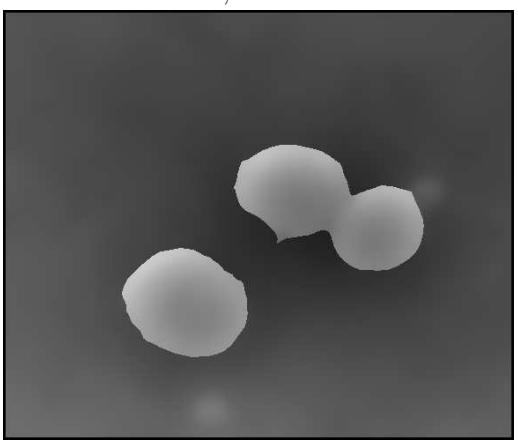
$k = 71,\ J = 0.0156$



$k = 91,\ J = 0.0160$

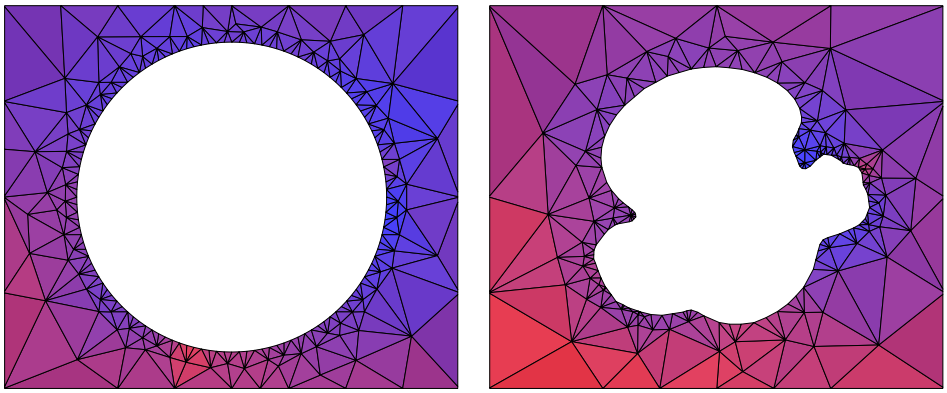


$k = 131,\ J = 0.0149$

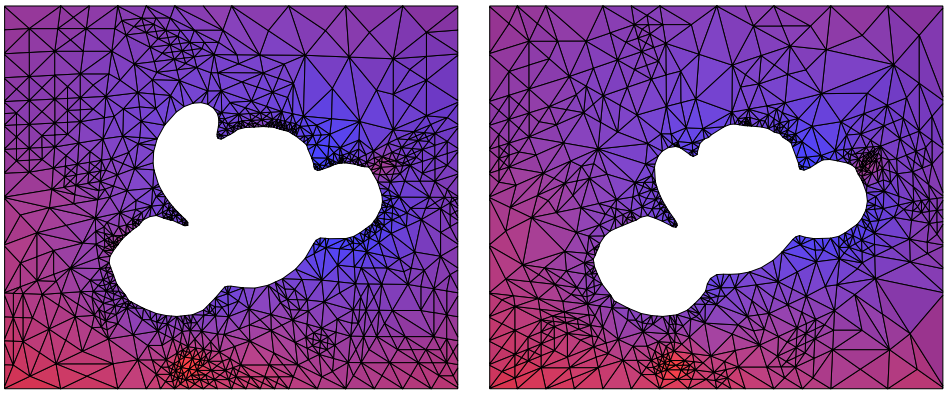


$k = 161,\ J = 0.0127$

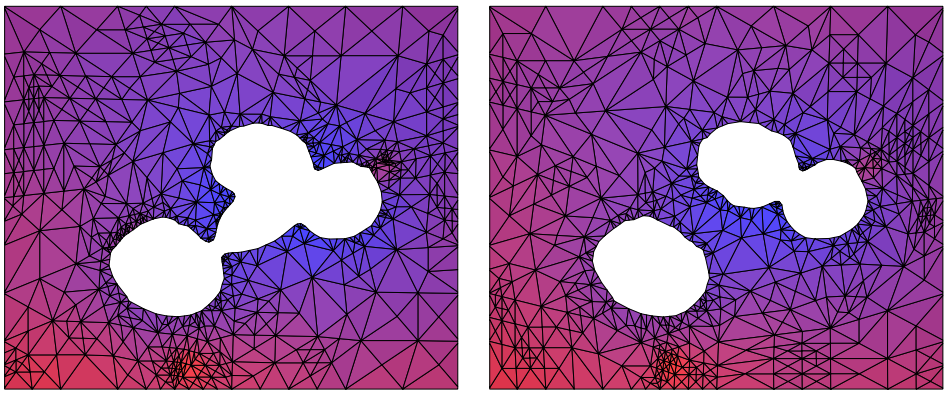
Figure 5.29: The evolution of the smooth approximations and the domains Ω_1 and Ω_2 for the bacteria image given by H^1 flow.



$k = 1$ $k = 121$

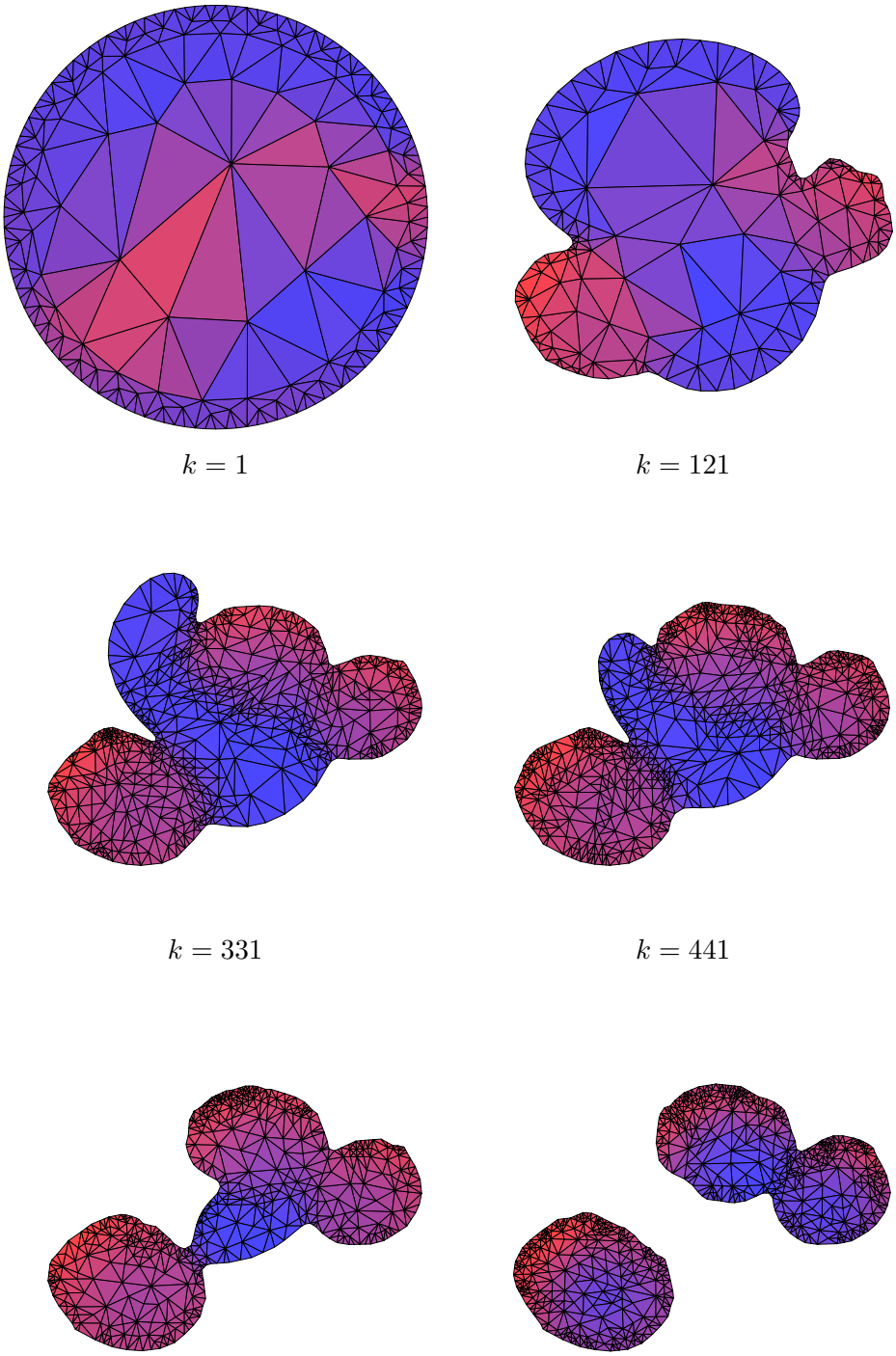


$k = 331$ $k = 441$



$k = 671$ $k = 841$

Figure 5.30: Snapshots of the background mesh for Ω_2 through iterations of the L^2 flow. Note that the mesh is locally refined in the last four images as adaptivity in domain is switched on after $N_{coarse} = 150$ coarse iterations.

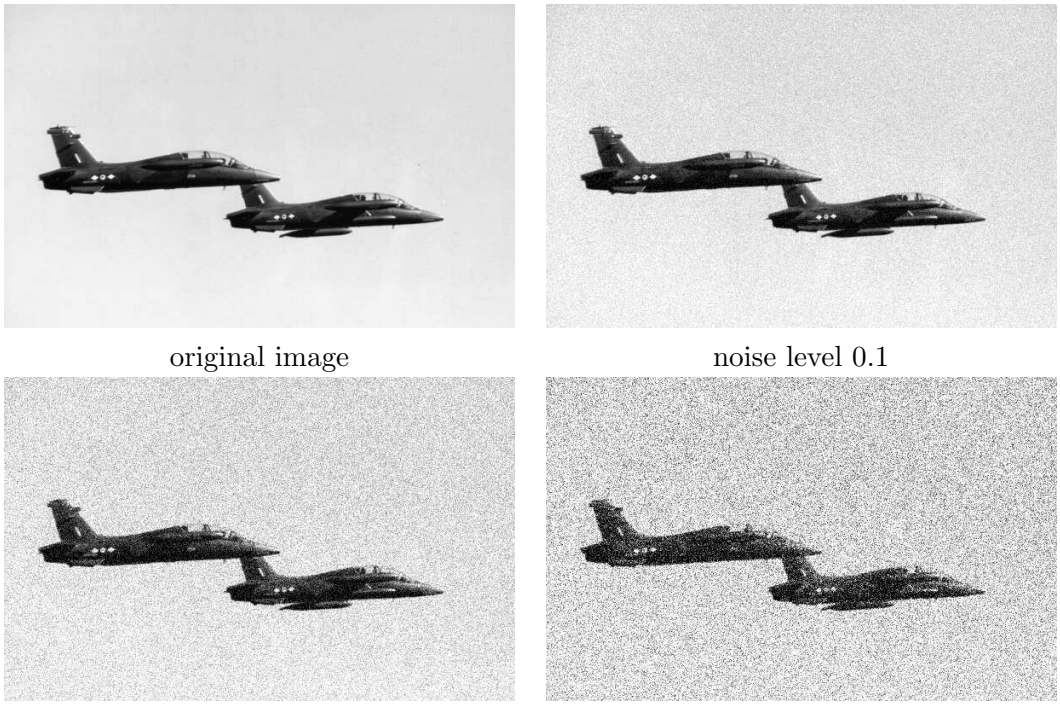


$k = 1$ $k = 121$

$k = 331$ $k = 441$

$k = 671$ $k = 841$

Figure 5.31: Snapshots of the foreground mesh for Ω_1 through iterations of the L^2 flow. Note in particular that after 150 iterations, space adaptivity is switched on and the mesh is refined at the object boundaries to capture the image function better. The mesh is coarser elsewhere.



original image noise level 0.1

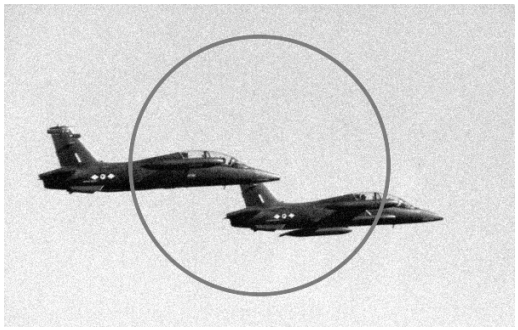
noise level 0.25 noise level 0.5

Figure 5.32: The original jet image and the sequence of noisy images at noise levels of 0.1, 0.25, 0.5 from top left to bottom right respectively.

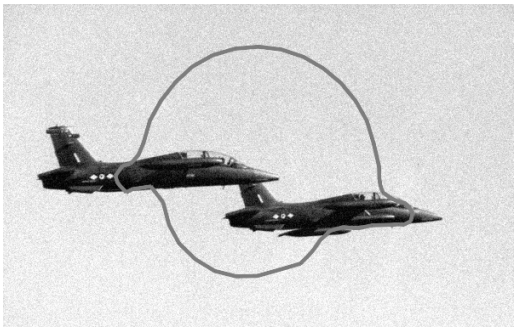
for different noise levels are shown in Figure 5.36. As the method slows down with increasing noise, we chose to take $N_{coarse} = 500$ coarse iterations for the the last image. Our interpretation for the decreasing performance is as follows. As the amount of noise increases, the background and the foreground become less and less dissimilar; therefore, it becomes more difficult for the method to obtain a proper separation.

5.2.3 Dependence on the Model Parameters

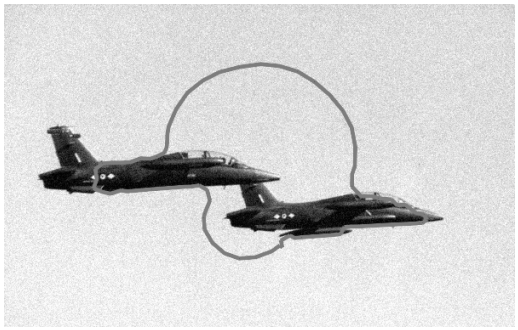
In this section we examine the sensitivity of the model (5.4) with respect to the model parameters μ and γ. Recall that μ is used to adjust the smoothness of the approximation to the image obtained with (5.4) whereas γ is a regularity parameter



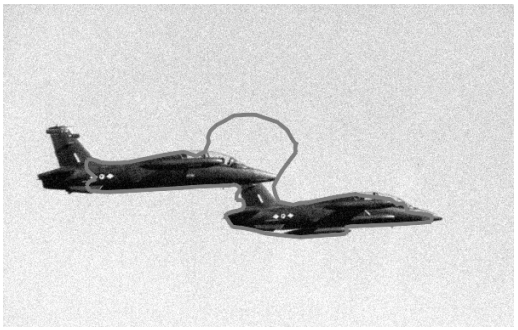
$k = 1,\ J = 0.0604$



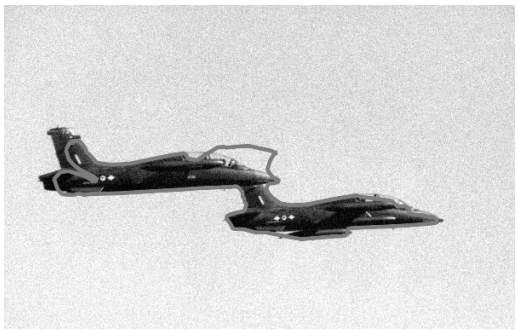
$k = 11,\ J = 0.0541$



$k = 21,\ J = 0.0422$



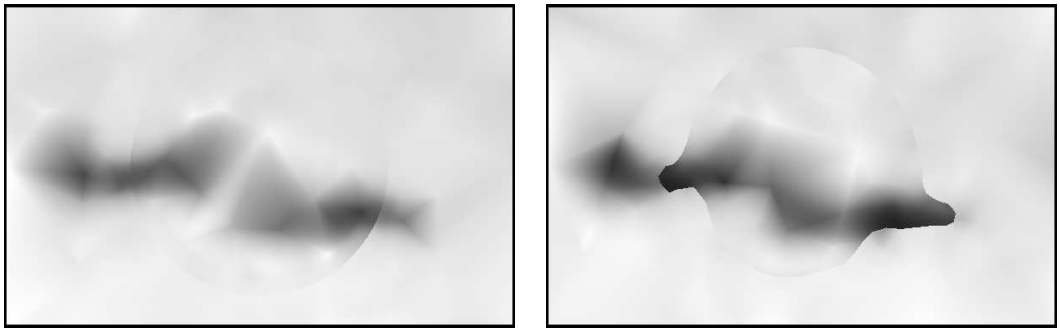
$k = 41,\ J = 0.0308$



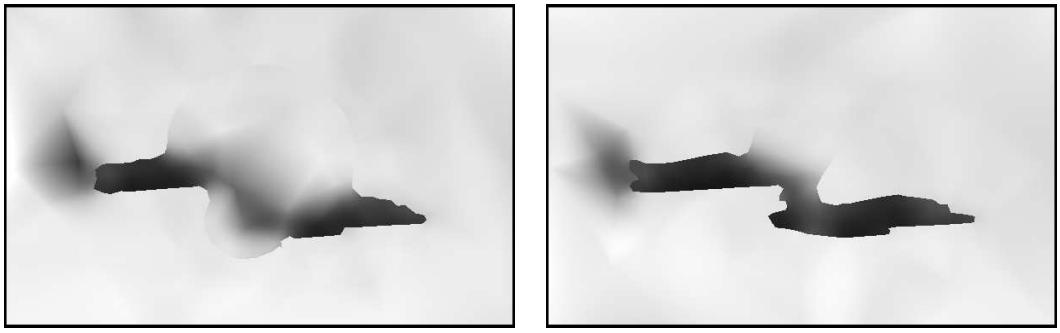
$k = 61,\ J = 0.0257$

$k = 171,\ J = 0.0197$

Figure 5.33: The evolution of the curve Γ given by H^1 flow for the noisy jet image with noise level 0.1.



$k = 1,\ J = 0.0604$ $k = 11,\ J = 0.0541$



$k = 21,\ J = 0.0422$ $k = 41,\ J = 0.0308$

$k = 61,\ J = 0.0257$ $k = 171,\ J = 0.0197$

Figure 5.34: The evolution of the piecewise smooth approximation and the domains Ω_1 and Ω_2 for the noisy jet image with noise level 0.1.

$k = 161$ $k = 161$

Figure 5.35: The results obtained for noise level 0.25 by using the same termination tolerance as for noise level 0.1. The method terminates prematurely.

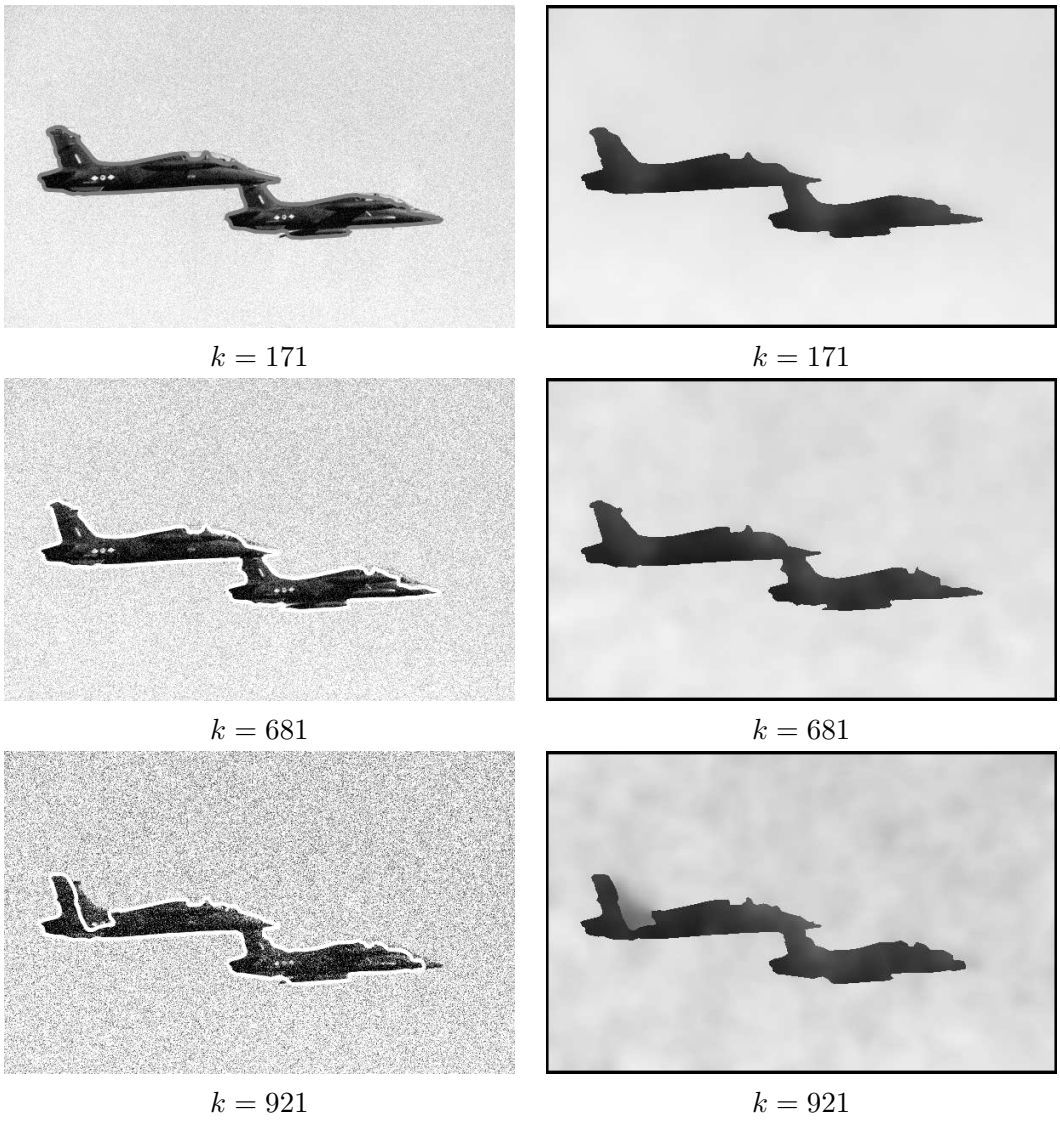


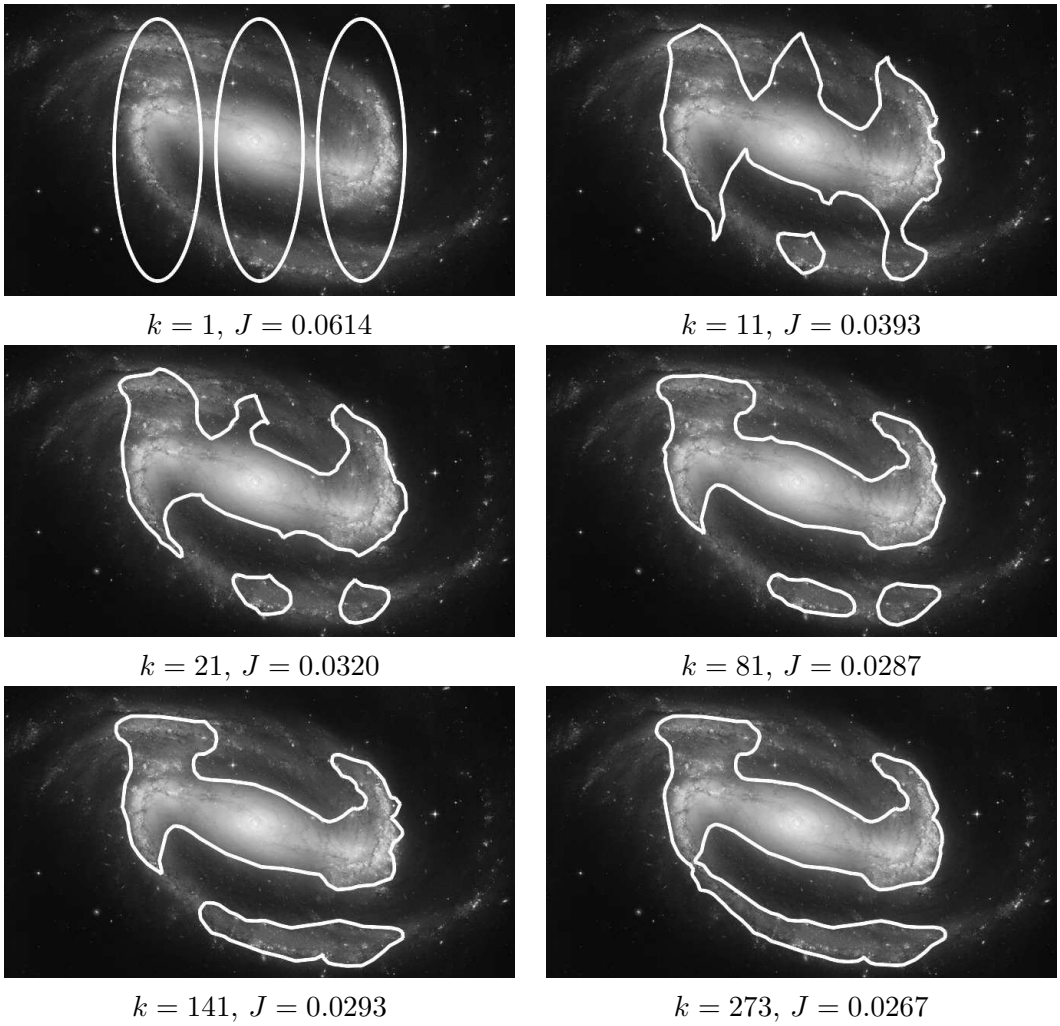
Figure 5.36: The results obtained for noise levels 0.1, 0.25, 0.5 from top to bottom. The left column shows the locations of the final curves and the right column shows the final smooth approximations. We see that as noise level increases, the method takes more iterations to terminate. We report k for each example.

for the curves and increasing γ results in shorter curves.

For the experiments, we use a galaxy image that we obtained from Wikimedia Commons (http://commons.wikimedia.org). This is a challenging example, because it does not have clear boundaries as in the previous examples. On the contrary, the intensity of the object gradually fades away to blend in with intensity of the background. Nevertheless it is possible to extract this object using the Mumford-Shah model (5.4), because the model utilizes global information of the image. The segmentation process is realized in the form of a pursuit to obtain a separation of the image into a background and a foreground given by two distinct smooth functions.

For the experiments we set the model parameters to be: $\mu = 0.2$, $\gamma = 5 \times 10^{-4}$; termination tolerance $tol = 5 \times 10^{-4}$; multilevel parameters: $N_{coarse} = 200$, $C_1 = 1, C_2 = 2$, $C_{coarse} = 3$, $d_1 = 5\,tol$, $d_2 = 20\,tol$; time steps: $\tau_1 = 0.02$, $\tau_2 = 0.04$, $\tau_{coarse} = 0.1$. We start the experiments with three oval initial curves. We see in Figure 5.37 that these quickly merge and split to capture a rough outline of the shape. Then they gradually converge to what we find to be an acceptable segmentation.

Next we explore the effect of changing the model parameters. We increase μ to 0.5, but this causes the initial curves to shrink and eventually vanish. The final approximation is a grayish image that is almost constant. When we decrease μ to 0.02, we get a segmentation that looks reasonable, but not as satisfying as the one obtained with $\mu = 0.2$. Still computationally the result is correct and the curves are at a minimum. We also see that $\mu = 0.02$ yields an approximation that is somewhat less smooth than those with higher μ values. These results are shown in Figure 5.39.



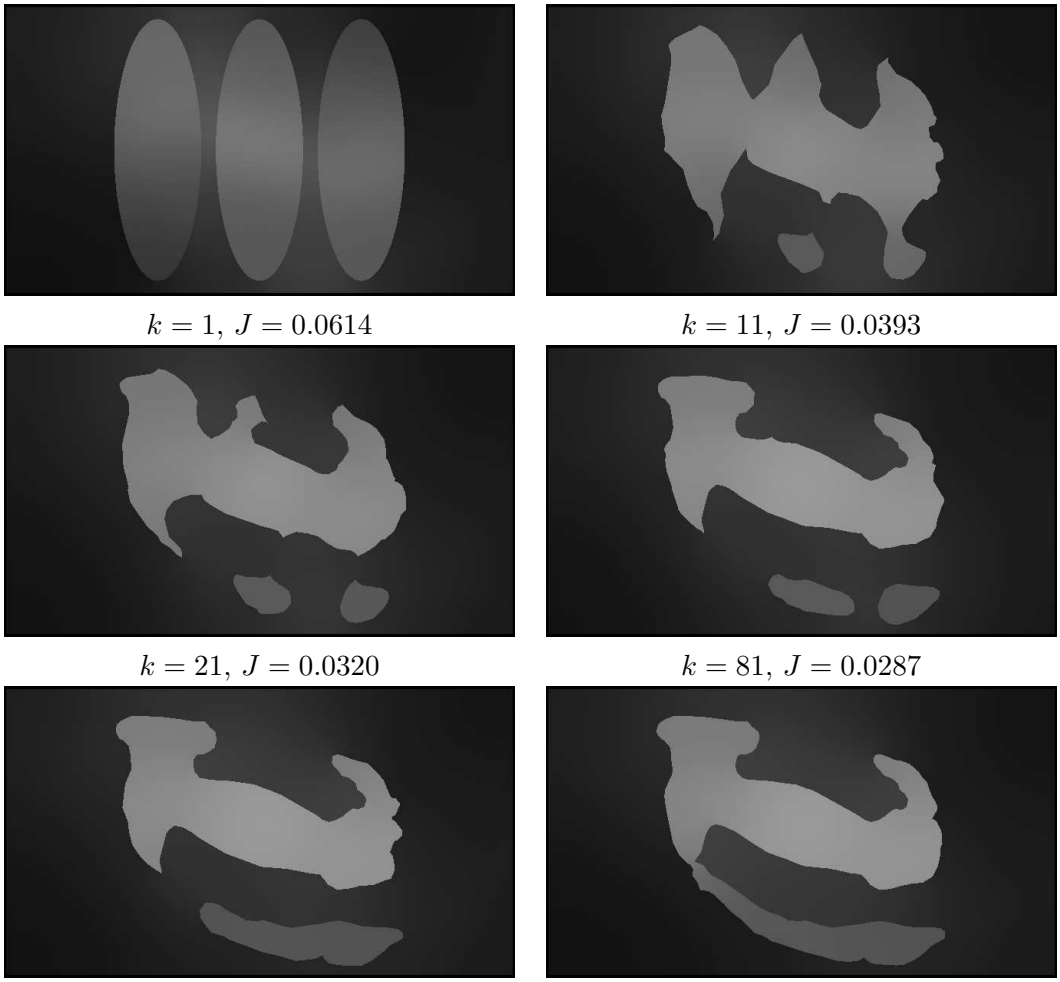
$k = 1,\ J = 0.0614$ $k = 11,\ J = 0.0393$

$k = 21,\ J = 0.0320$ $k = 81,\ J = 0.0287$

$k = 141,\ J = 0.0293$ $k = 273,\ J = 0.0267$

Figure 5.37: The evolution of the curve Γ given by H^1 flow for the galaxy image. The parameters of the model are: $\mu = 0.2$, $\gamma = 5 \times 10^{-4}$.

Finally we execute a set of experiments varying the value of γ. We increase γ to 0.002 and see that this causes the small curves at the bottom of the image to vanish. Therefore, we are only able to capture the salient core of the galaxy. When we decrease γ to 2×10^{-4}, we obtain the previous result for $\gamma = 5 \times 10^{-4}$ plus another strip at the top. We also see that the final curves are not as smooth. This is because by using a smaller value for γ, we relax the length penalty on the curve and it has more freedom to fit local variations. These results are given in Figure 5.40.

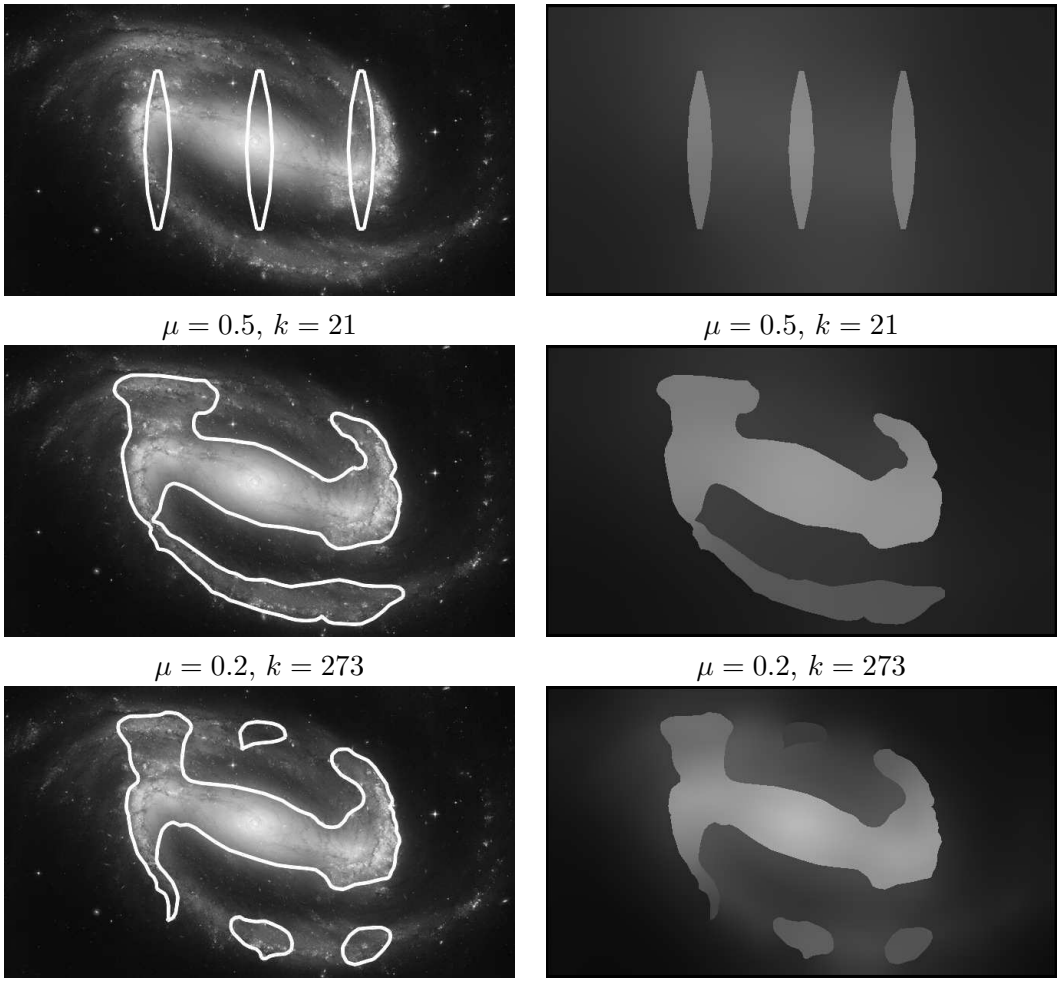


$k = 1,\ J = 0.0614$ $k = 11,\ J = 0.0393$

$k = 21,\ J = 0.0320$ $k = 81,\ J = 0.0287$

$k = 141,\ J = 0.0293$ $k = 273,\ J = 0.0267$

Figure 5.38: The evolution of the smooth approximation and domains Ω_1, Ω_2 to the galaxy image given by H^1 flow. The parameters of the model are: $\mu = 0.2$, $\gamma = 5 \times 10^{-4}$.

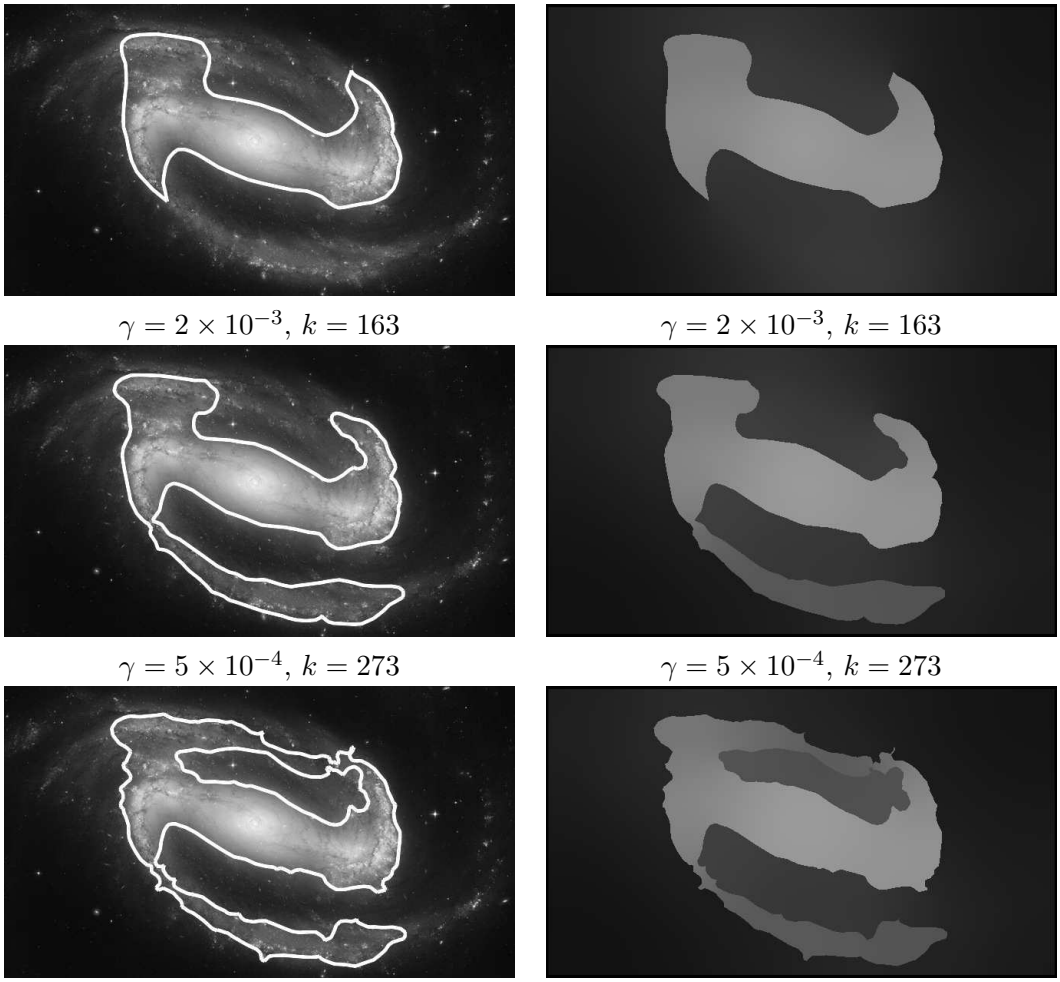


$\mu = 0.5$, $k = 21$ $\mu = 0.5$, $k = 21$

$\mu = 0.2$, $k = 273$ $\mu = 0.2$, $k = 273$

$\mu = 0.02$, $k = 149$ $\mu = 0.02$, $k = 149$

Figure 5.39: The segmentation results obtained by varying the parameter μ. The curves shrink and disappear completely when we increase μ to 0.5 (see top row). Decreasing μ to 0.02 still gives a reasonable result (see bottom row). We can also see that the approximations get somewhat less smooth as we decrease μ. For the highest value the final approximation is almost a constant gray value.



$\gamma = 2 \times 10^{-3}$, $k = 163$ $\gamma = 2 \times 10^{-3}$, $k = 163$

$\gamma = 5 \times 10^{-4}$, $k = 273$ $\gamma = 5 \times 10^{-4}$, $k = 273$

$\gamma = 2 \times 10^{-4}$, $k = 301$ $\gamma = 2 \times 10^{-4}$, $k = 301$

Figure 5.40: The segmentation results obtained by varying the parameter γ. These experiment clearly show how the length term in the Mumford-Shah model constrain the curves. For the highest value of the parameter, $\gamma = 2 \times 10^{-3}$, we obtain shorter and smoother curves. For the lowest value, $\gamma = 2 \times 10^{-4}$, we obtain longer curves that are much less smooth.

5.2.4 Choice of Initial Curves

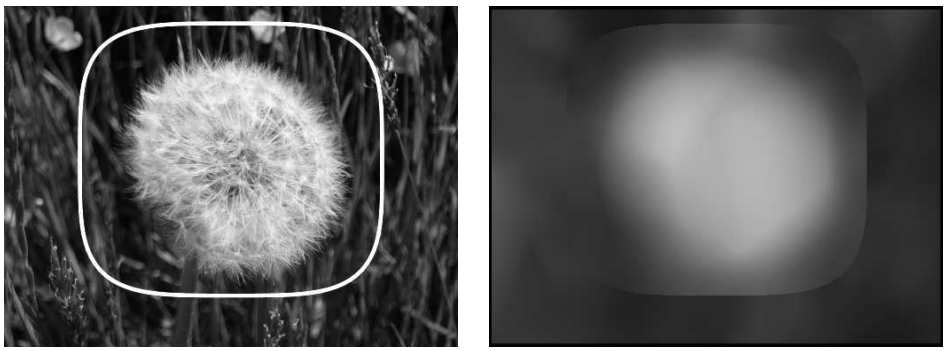
In this section we examine the sensitivity of the solution with respect to the choice of initial curves. We run our tests on the image of a single dandelion with a cluttered background. The image is from Wikipedia (http://www.wikipedia.org). The parameters of the model are: $\mu = 0.02$, $\nu = 0.001$; termination tolerance $tol = 0.001$; multilevel parameters: $N_{coarse} = 100$, $C_1 = 1, C_2 = 2$, $C_{coarse} = 3$, $d_1 = 5\,tol$, $d_2 = 20\,tol$; time steps $\tau_1 = 0.02$, $\tau_2 = 0.04$, $\tau_{coarse} = 0.1$.

As the Mumford-Shah energy (5.4) is nonconvex, one cannot expect strict independence of the solution from the initial curve. Nevertheless in our experiments we observe a certain degree of robustness with respect to the initial curve. We document this with three experiments. In the first experiment we start with an initial curve that contains the dandelion. The curve shrinks and captures the object (see Figure 5.41). In the second experiment, we start with a curve that is completely inside the object. The curve expands outward and captures the boundary see (Figure 5.42). In the third experiment, we start with a curve that partially overlaps with the object. We see that it is sucked inside to end up at the boundary of the object (see Figure 5.43).

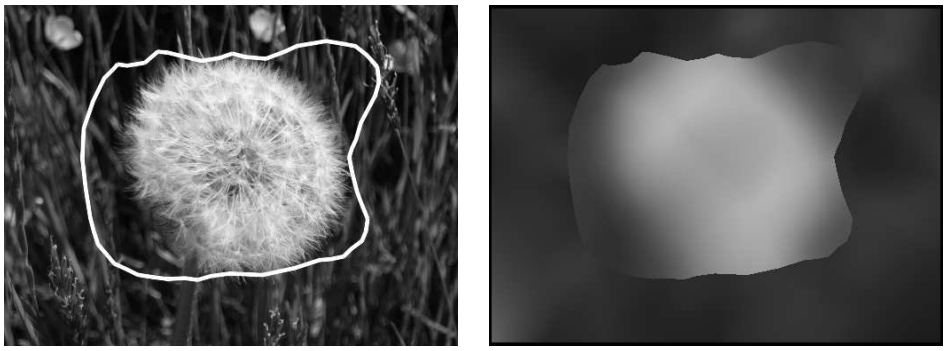
One point we should emphasize is that we did not modify the model parameters to obtain these various results. This is in contrast with the minimal surface model, where we had to adjust the coefficient γ of the domain term to get a shrinking or expanding evolution. We should caution however that this is not supposed to be behavior under all circumstances. For example, if we choose γ large enough in (5.4),

it becomes very hard to obtain expanding evolutions.

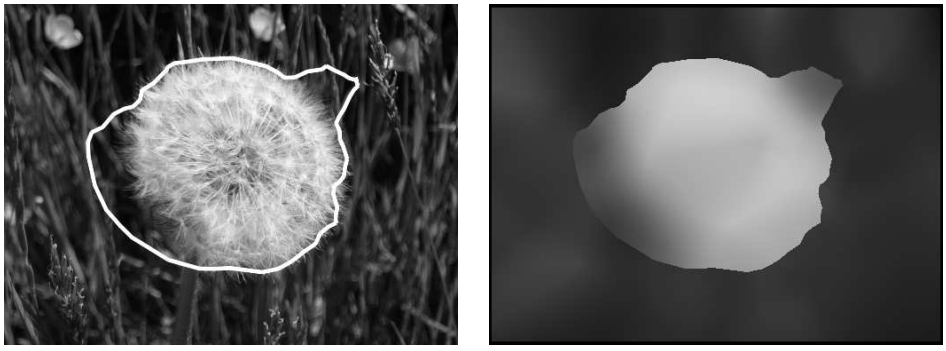
Our last example once again reveals us the nonconvex nature of this problem. We start with an initial curve inside the object. But in contrast to the second example, the curve does not result in a separation of only foreground and background (see Figure 5.44). It separates the dandelion into two regions as well. As much as this might look wrong with regard to what we expect. This result is just fine from a shape optimization standpoint. The resulting configuration corresponds to a local minimum; in particular, the shape derivative is below the tolerance.



$k = 1,\ J = 0.0673$ $k = 1,\ J = 0.0673$



$k = 11,\ J = 0.0520$ $k = 11,\ J = 0.0520$



$k = 31,\ J = 0.0404$ $k = 31,\ J = 0.0404$

$k = 100,\ J = 0.0281$ $k = 100,\ J = 0.0281$

Figure 5.41: The evolution of the curve given by H^1 flow for the dandelion image. The initial curve is outside the object.

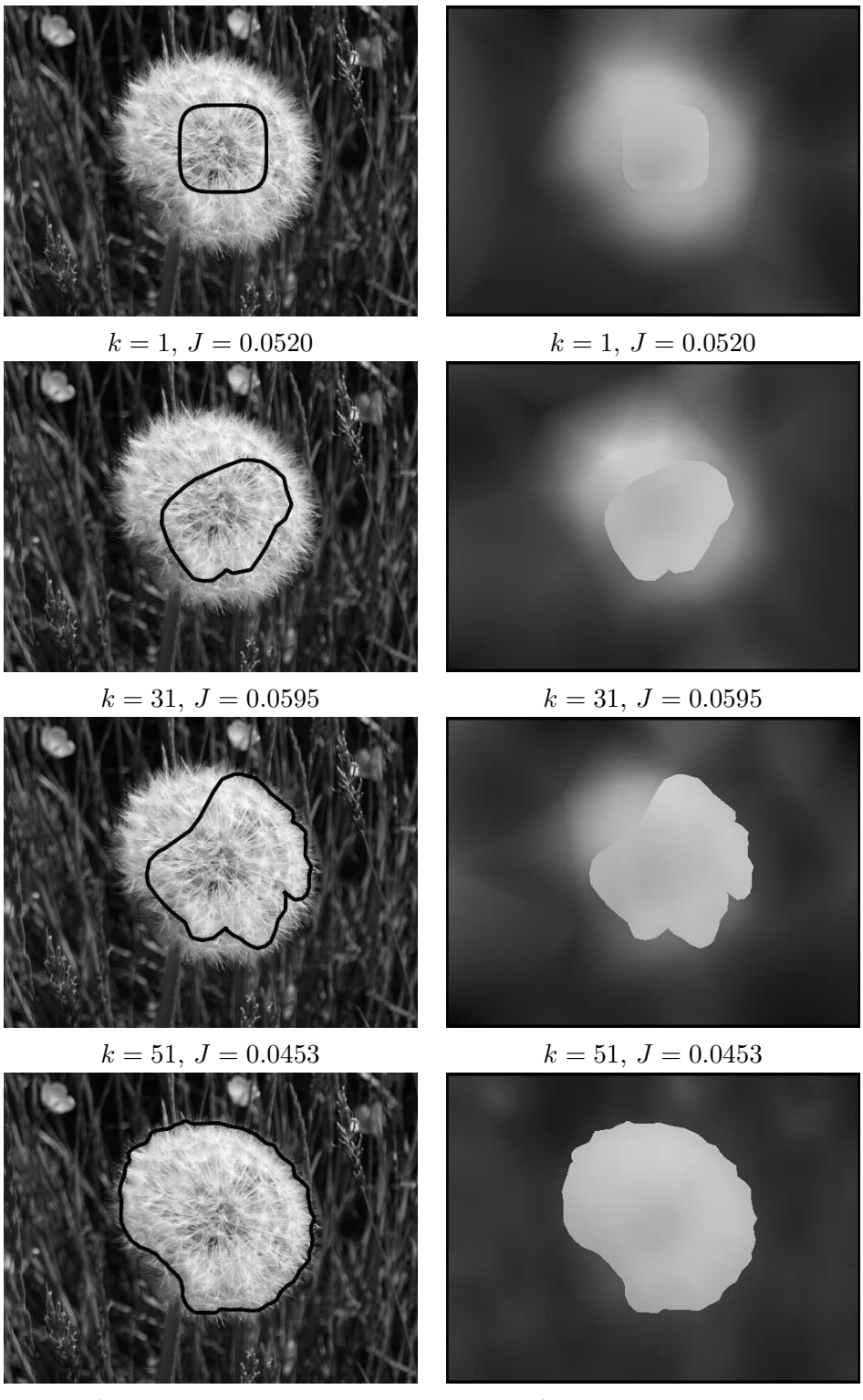
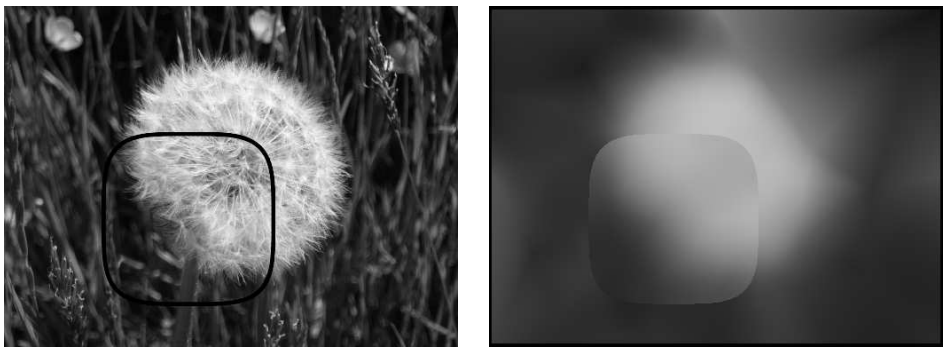
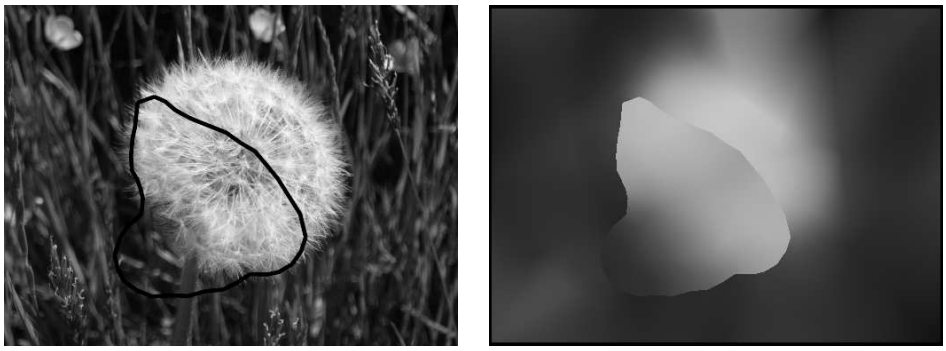


Figure 5.42: The evolution of the curve given by H^1 flow for the dandelion image. The initial curve is inside the object.



$k = 1,\ J = 0.0635$ $k = 1,\ J = 0.0635$

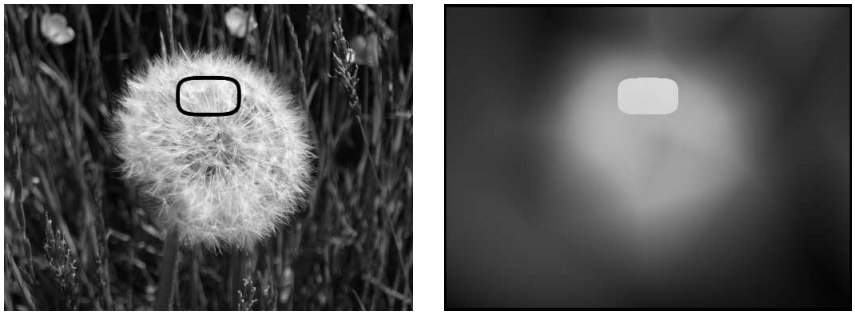


$k = 11,\ J = 0.0544$ $k = 11,\ J = 0.0544$

$k = 31,\ J = 0.0440$ $k = 31,\ J = 0.0440$

$k = 113,\ J = 0.0339$ $k = 113,\ J = 0.0339$

Figure 5.43: The evolution of the curve given by H^1 flow for the dandelion image. The initial curve partially overlaps with the object.



$k = 1,\ J = 0.0577$ $k = 1,\ J = 0.0577$

$k = 41,\ J = 0.0404$ $k = 41,\ J = 0.0404$

$k = 91,\ J = 0.0338$ $k = 91,\ J = 0.0338$

$k = 210,\ J = 0.0353$ $k = 210,\ J = 0.0353$

Figure 5.44: The evolution of the curve given by H^1 flow for the dandelion image. The initial curve is inside the object. Notice that the energy at $k = 91$ is lower than that at $k = 210$. What happens is that the curve moves away from the local minimum around $k = 91$ with the coarse iterations since there is no check on the energy or the shape derivative during the coarse phase. Eventually the iterations terminate at another local minimum at $k = 210$.

Bibliography

[1] G. Allaire, F. Jouve, and A.-M. Toader. Structural optimization using sensitivity analysis and a level-set method. *J. Comput. Phys.*, 194(1):363–393, 2004.

[2] G. Allaire and R. V. Kohn. Optimal design for minimum weight and compliance in plane stress using extremal microstructures. *European J. Mech. A Solids*, 12(6):839–878, 1993.

[3] F. Almgren. Questions and answers about area-minimizing surfaces and geometric measure theory. In *Differential geometry: partial differential equations on manifolds (Los Angeles, CA, 1990)*, volume 54 of *Proc. Sympos. Pure Math.*, pages 29–53. Amer. Math. Soc., Providence, RI, 1993.

[4] F. Almgren and J. E. Taylor. Optimal geometry in equilibrium and growth. *Fractals*, 3(4):713–723, 1995. Symposium in Honor of Benoit Mandelbrot (Curaçao, 1995).

[5] F. Almgren and J. E. Taylor. Soap bubble clusters. *Forma*, 11(3):199–207, 1996.

[6] L. Ambrosio and V. M. Tortorelli. On the approximation of free discontinuity problems. *Boll. Un. Mat. Ital. B (7)*, 6(1):105–123, 1992.

[7] K. E. Atkinson. *An introduction to numerical analysis*. John Wiley & Sons Inc., New York, second edition, 1989.

[8] G. Aubert, M. Barlaud, O. Faugeras, and S. Jehan-Besson. Image segmentation using active contours: calculus of variations or shape gradients? *SIAM J. Appl. Math.*, 63(6):2128–2154 (electronic), 2003.

[9] D. Azé and G. Buttazzo. Some remarks on optimal design of periodically reinforced structures. *RAIRO Modeél. Math. Anal. Numér.*, 23(1):53–61, 1989.

[10] E. Bänsch. Finite element discretization of the Navier-Stokes equations with a free capillary surface. *Numer. Math.*, 88(2):203–235, 2001.

[11] E. Bänsch, P. Morin, and R. Nochetto. A finite element method for surface diffusion: the parametric case. *J. Comput. Phys.*, 203(1):321–343, 2005.

[12] B. Bourdin and A. Chambolle. Implementation of an adaptive finite-element approximation of the Mumford-Shah functional. *Numer. Math.*, 85(4):609–646, 2000.

[13] D. Bucur and G. Buttazzo. *Variational methods in shape optimization problems*. Progress in Nonlinear Differential Equations and their Applications, 65. Birkhäuser Boston Inc., Boston, MA, 2005.

[14] D. Bucur, G. Buttazzo, and A. Henrot. Existence results for some optimal partition problems. *Adv. Math. Sci. Appl.*, 8(2):571–579, 1998.

[15] M. Burger. A framework for the construction of level set methods for shape optimization and reconstruction. *Interfaces Free Bound.*, 5(3):301–329, 2003.

[16] M. Burger and S. J. Osher. A survey on level set methods for inverse problems and optimal design. *European J. Appl. Math.*, 16(2):263–301, 2005.

[17] G. Buttazzo and G. Dal Maso. An existence result for a class of shape optimization problems. *Arch. Rational Mech. Anal.*, 122(2):183–195, 1993.

[18] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *IJCV*, 22(1):61–79, 1997.

[19] V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert. Minimal surfaces: a geometric three-dimensional segmentation approach. *Numer. Math.*, 77(4):423–451, 1997.

[20] T. Chan and L. Vese. A level set algorithm for minimizing the mumford-shah functional in image processing. In *Proceedings of the 1st IEEE Workshop on "Variational and Level Set Methods in Computer Vision"*, pages 161–168, 2001.

[21] T. F. Chan and L. A. Vese. Active contours without edges. *Image Processing, IEEE Transactions on*, 10(2):266–277, 2001.

[22] D. Chenais. On the existence of a solution in a domain identification problem. *J. Math. Anal. Appl.*, 52(2):189–219, 1975.

[23] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[24] K. Deckelnick, G. Dziuk, and C. M. Elliott. Computation of geometric partial differential equations and mean curvature flow. *Acta Numer.*, 14:139–232, 2005.

[25] M. C. Delfour and J.-P. Zolésio. *Shapes and Geometries*. Advances in Design and Control. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.

[26] H. Delingette and J. Montagnat. Shape and topology constraints on parametric active contours. *Computer Vision and Image Understanding*, 83(2):140–171, 2001.

[27] A. Desolneux, L. Moisan, and J.-M. Morel. Variational snake theory. In *Geometric level set methods in imaging, vision, and graphics*, pages 79–99. Springer, New York, 2003.

[28] G. Doğan, P. Morin, R. H. Nochetto, and M. Verani. Discrete gradient flows for shape optimization and applications. *submitted*.

[29] G. Dziuk. An algorithm for evolutionary surfaces. *Numer. Math.*, 58(6):603–611, 1991.

[30] O. Faugeras and R. Keriven. Variational principles, surface evolution, pde's, level set methods and the stereo problem. In *IEEE Trans. Image Processing*, volume 7, pages 336–344, 1998.

[31] P. Fua and Y. G. Leclerc. Model driven edge detection. *Machine Vision and Applications*, 3:45–56, 1990.

[32] M. P. Glotzbecker, D. F. Carpentieri, and J. P. Dormans. Langerhans cell histiocytosis: Clinical presentation, pathogenesis, and treatment from the lch etiology research group at the children's hospital of philadelphia. *UPOJ*, 15:67–73, 2002.

[33] J. Haslinger and R. A. E. Mäkinen. *Introduction to shape optimization*, volume 7 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2003. Theory, approximation, and computation.

[34] A. Henrot and M. Pierre. *Variation et Optimisation de Formes*, volume 48 of *Mathématiques et Applications*. Springer, 2005.

[35] M. Hintermüller and W. Ring. A second order shape optimization approach for image segmentation. *SIAM J. Appl. Math.*, 64(2):442–467, 2003/04.

[36] M. Hintermüller and W. Ring. An inexact Newton-CG-type active contour approach for the minimization of the Mumford-Shah functional. *J. Math. Imaging Vision*, 20(1-2):19–42, 2004. Special issue on mathematics and image analysis.

[37] J. T. Jenkins. The equations of mechanical equilibrium of a model membrane. *SIAM J. Appl. Math.*, 32(4):755–764, 1977.

[38] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, 1988.

[39] R. Kimmel and A. Bruckstein. Regularized laplacian zero crossings as optimal edge integrators. *IJCV*, 53(3):225–243, 2003.

[40] J.-O. Lachaud and A. Montanvert. Deformable meshes with automated topology changes for coarse-to-fine 3D surface extraction. *Medical Image Analysis*, 3(2):187–207, 1999.

[41] E. Laporte and P. Le Tallec. *Numerical methods in sensitivity analysis and shape optimization*. Modelling and simulation in science, engineering and technology. Birkhüser, Boston, MA, 2002.

[42] M. Leventon, O. Faugeraus, and W. Grimson. Level set based segmentation with intensity and curvature priors. In *Proceedings of Workshop on Mathematical Methods in Biomedical Image Analysis Proceedings*, pages 4–11, 2000.

[43] S. Luckhaus. Solutions for the two-phase Stefan problem with the Gibbs-Thomson law for the melting temperature. *European J. Appl. Math.*, 1(2):101–111, 1990.

[44] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *MIA*, 1(2):91–108, 1996.

[45] T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000.

[46] A. Miele. *Theory of optimum aerodynamic shapes*. Academic Press, New York, 1965.

[47] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Numerical Mathematics and Scientific Computation. The Clarendon Press Oxford University Press, New York, 2001. Oxford Science Publications.

[48] J. Montagnat, H. Delingette, and N. Ayache. A review of deformable surfaces: topology, geometry and deformation. *Image and Vision Computing*, 19(14):1023–1040, December 2001.

[49] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math.*, 42(5):577–685, 1989.

[50] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999.

[51] A. Novruzi and M. Pierre. Structure of shape derivatives. *J. Evol. Equ.*, 2(3):365–382, 2002.

[52] A. Novruzi and J. R. Roche. Newton's method in shape optimisation: a three-dimensional case. *BIT*, 40(1):102–120, 2000.

[53] S. Osher and R. Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2003.

[54] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[55] S. J. Osher and F. Santosa. Level set methods for optimization problems involving geometry and constraints. I. Frequencies of a two-density inhomogeneous drum. *J. Comput. Phys.*, 171(1):272–288, 2001.

[56] N. Paragios and R. Deriche. Coupled geodesic active regions for image segmentation: A level set approach. In *ECCV (2)*, pages 224–240, 2000.

[57] O. Pironneau. *Optimal Shape Design for Elliptic Systems*. Springer Series in Computational Physics. Springer-Verlag, New York, 1984.

[58] C. Ramananjaona, M. Lambert, D. Lesselier, and J.-P. Zolésio. Shape reconstruction of buried obstacles by controlled evolution of a level set: from a min-max formulation to numerical experimentation. *Inverse Problems*, 17(4):1087–1111, 2001. Special issue to celebrate Pierre Sabatier's 65th birthday (Montpellier, 2000).

[59] M. Rousson, T. Brox, and R. Deriche. Active unsupervised texture segmentation on a diffusion based feature space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 699–704, Madison, WI, June 2003.

[60] A. Schmidt and K. Siebert. *Design of Adaptive Finite Element Software*, volume 42 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2005. The finite element toolbox ALBERTA, With 1 CD-ROM (Unix/Linux).

[61] J. A. Sethian. *Level set methods and fast marching methods*, volume 3 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, second edition, 1999. Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science.

[62] J. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In M. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[63] A. Singh, D. Terzopoulos, and D. B. Goldgof. *Deformable Models in Medical Image Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1998.

[64] J. Sokołowski and J.-P. Zolésio. *Introduction to Shape Optimization*, volume 16 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1992. Shape sensitivity analysis.

[65] J. E. Taylor, editor. *Computational Crystal Growers Workshop*, Selected Lectures in Mathematics, Providence, RI, 1992. American Mathematical Society.

[66] J. E. Taylor, J. W. Cahn, and C. A. Handwerker. Geometric models of crystal growth. *Acta Metall. Mater.*, 40:1443–1474, 1992.

[67] A. Tsai, A. Yezzi, and A. Willsky. Curve evolution implementation of the Mumford-Shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Transactions on Image Processing*, 10(8):1169–1186, 2001.

[68] T. J. Willmore. *Total curvature in Riemannian geometry*. Ellis Horwood Series: Mathematics and its Applications. Ellis Horwood Ltd., Chichester, 1982.

[69] J. Zhang. Level set method for shape optimization of plate piezoelectric patches. *Methods Appl. Anal.*, 10(2):329–346, 2003.