2007

This is to certify that the
dissertation entitled

INTERPOLATION SNAKES WITH SHAPE PRIOR FOR
BOUNDARY DETECTION IN NOISY IMAGES

presented by

SILVIU D. MINUT

has been accepted towards fulfillment
of the requirements for the

Ph. D.	degree in	COMPUTER SCIENCE AND ENGINEERING

[illegible]
Major Professor's Signature

22 Dec 2006

Date

Interpolation Snakes with Shape Prior for Boundary Detection in Noisy Images

By

Silviu D. Minut

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2007

Abstract

Interpolation Snakes with Shape Prior for Boundary Detection in Noisy Images

By

Silviu D. Minut

We develop an active contour algorithm (I-SNAKE) designed to cope well with the noise, specularity and incomplete visual support typically found in ultrasound images. To impose smoothness, we model mathematically our active contour as an interpolation spline. This model also allows a user to set hard spatial constraints on the snake by fixing one or more control points of the spline. We introduce a shape prior term in the snake energy functional that acts as shape memory and prevents excessive deviations from a known shape model computed from shape training samples. This shape model also supports operation with occlusion and loss of contrast. Extensive tests on 117 ultrasound images comparing the similarity of the detected boundaries to manually traced contours (ground truth) showed that the interpolation snake algorithm (I-SNAKE) did significantly better than three other active contour algorithms: the original Kass-Witkin-Terzopoulos snakes (KWT), the Geodesic Active Contour model (GAC) and the Chan-Vese model (CV). Moreover, for clinical validity a subjective evaluation by a human expert was performed via a blind study that compared I-SNAKE with the ground truth (GT) or with the KWT model. The I-SNAKE boundary was preferred in 80 out of 117 comparisons with GT, and in 99 out of 117 with KWT. Other example results are given in several different image domains.

To Ana and Aurelia

ACKNOWLEDGMENTS

First and foremost I would like to express my deep gratitude and respect to my adviser, Dr. George C. Stockman - a man of great stature, patience and understanding and a true gentleman, who supported me in more ways than one. Fervent academic discussions contributed to my understanding of different aspects of computer vision, and led to a number of ideas materialized in this thesis. Literally, this document would not exist without the support of Dr. Stockman.

I also want to thank the other members of the committee, Dr. John Weng, Dr. Robert McGough and Dr. Charles MacCluer for their invaluable expertise and for their unconditional support. To me, this thesis marks the end of one period in my life, but more importantly, it represents a new beginning for the many research projects that have yet to be pursued. Stronger than the words that I can find for expressing my gratitude is the promise and the hope for future collaboration on these projects, with all the committee members.

I am grateful to the faculty and student members of the IGERT program in Cognitive Science at MSU, for the many challenging and intellectually stimulating discussions. I am particularly indebted to Dr. Fred C. Dyer from the Zoology Department at Michigan State University and to Dr. John M. Henderson from the Department of Psychology (now with the University of Edinburgh) - two men with a vision, and with broad scientific horizons, who offered me financial support under the IGERT grant, and who helped me see well beyond computer vision.

I am grateful to Drs. Bari Olivier and Augusta Pelosi, from the Department of Small Animal Clinical Sciences at MSU for providing a database of cardiac ultrasound images.

I want to acknowledge Kitware Inc. - the producer of the Insight Toolkit (ITK) - for making available their wonderful, open-source software. I used their code on several occasions in this thesis.

Special thanks go to Dr. Dorry Segev, from the Division of Transplantation, Department of Surgery at Johns Hopkins Medical Institutions and to his wife Dr. Sommer Gentry, from the U.S. Naval Academy, for their support and generosity. Dr. Segev took an interest in my work, and provided me with a substantial database of kidney ultrasound images - the centerpiece of the entire thesis. Furthermore, Dr. Segev took invaluable time in manually tracing the kidney contours in all images in the database - the basis for the evaluation process of the quality of my work. Dr. Segev graciously agreed, and took part as a judge in a blind study of the performance of several border detection algorithms - an experiment which, hopefully, will confer clinical validity to my work. In short, Dr. Segev provided the raw material for this thesis, and his medical expertise, without which this thesis would have not been completed. His recognition of the results brings the hope that the ideas in this thesis may materialize into a real world application and gives meaning to my work.

Last, but not least, I must thank my wife for standing by my side for better and for worse, for putting up with a mad scientist all this time, while raising a family and developing her own career at the US Naval Academy. Her achievements far exceed mine, and I can only hope that the completion of this thesis will enable me to do all the things that she does.

Table of Contents

List of Tables

List of Figures

Images in this dissertation are presented in color.

Chapter 1

Introduction

1.1 To model, or not to model: A personal perspective

In order to find their way in the world, most - if not all - organisms must sense the environment around them. Most species try to "make sense" [1] of the world in their own way, enough to ensure their survival, and limited (sometimes severely) by the complexity of their nervous system.

We, humans, probably like many other species, parse the world into physically and functionally distinct entities. We mindlessly refer to these entities as *objects*. We use nouns like "apple", "tree", "heart", etc. to label these objects. But what is an object? Of course, in our daily life, we do just fine without a definition of the notion of "object", or even without a definition of most of the natural objects: we see *it*, and we can name it effortlessly and instantly, and we can do so from a very young age too. However, the moment we try to build intelligent artificial systems that are able to act in the world autonomously, in a human-like fashion, we *must* define or

[1]We use this term here without reference to controversial concepts like "mind", "consciousness", or "cognition".

represent objects in some precise, mathematical way, so that the computer can act on them.

Sometimes a human refers to things *generically* ("car", "house", etc.), and other times *specifically* ("this car", "that house"). Although largely unconscious, this distinction is crucial in trying to formalize in any useful way any concrete concept in the "mind" of the computer. "Car" is an abstraction, a generalization; "this car" is a particular instance which has roughly all *features* that describe a car, and possibly certain attributes or details that distinguish it from "another car". Although in daily life we loosely categorize both the abstraction and any of its instances as "objects", in pattern recognition, the term *class* is used to refer to a generic category of objects.

Let us also note that we can also readily sketch on paper (from memory) just about any of the categories of objects that we normally perceive. But, since a category of objects is a collection of many particular exemplars, what exactly is it that we are drawing? However little artistic talent we may posses, a drawing of an apple would never be mistaken for a boat, say. It appears then, that our brain carries some sort of representation, some sort of mental image (acquired from experience) of the various categories of (simple) objects. This representation somehow summarizes the essential characteristics of each of these categories. Whatever is being drawn on paper (or just imagined) is some sort of average, something representative for the entire class - in other words, a *model.*

So far, we have argued that humans parse the world into (classes) of objects, and build models of them. To be sure, we are far from claiming that the human brain builds a model of the *entire* world (at once), nor do we claim that the brain builds an internal panoramic image of the field of view, and acts upon that, as it was proposed in David Marr's theory of vision [72]. Studies on visual cognition, eye-movement and visual attention [18, 67, 106, 91, 3, 105, 48, 75, 76, 56, 55] (to name a few) have un-ambiguously dismantled these ideas, but do not exclude the possibility that the human brain does utilize models of individual objects.

We need to understand then, how to define (models of) classes and/or objects, for the purpose of using them on an intelligent machine. With few notable exceptions (such as some geometric objects like circle, line, etc. that can be described by equations), we, humans, commonly use different *attributes* or *features* to describe objects. Naturally, we can try to port these features to computer world, which entails that each of them must be quantifiable mathematically, so it can be measured and represented as a number. Thus, for instance, size attributes (such as length, width, volume), or color, curvature, etc. make for good feature candidates, while "beauty", "grace" or "malice" do not.

Precisely which features should be selected for describing one concept or another is the problem of *feature selection.* In traditional *pattern recognition* [33, 98, 57, 41], the programmer selects, often subjectively, whatever features seem to make more sense for the problem at hand, in effect trying to make the computer "perceive" the world the way *we* do. A system built in this manner is often termed *knowledge-based.* This is but one, heavily criticized, but still very popular way of building artificial intelligence (AI). It is by no means the only, or the best way. Most notably, one could mention *genetic algorithms* [50, 44], *reinforcement learning* [61, 101] or *developmental learning* [108, 109].

The point we are trying to make here is that we see the need for building computational models for AI systems. Based on our personal experience, we think that while the knowledge-based approach may not be powerful enough (as many people argue) to solve "the vision problem"[2], it is, at least for the time being, the most straightforward and the most reliable way of building an expert system, such as a face recognizer, or an object detector.

[2]Legend has it that some 50 years ago, Marvin Minsky assigned "the vision problem" to an undergraduate student, as a summer project.

1.2 Contributions and Overview of the Thesis

Throughout the thesis, our research is confined exclusively to 2D shapes, so unless otherwise specified, by "shape" we mean "2D shape".

Our main problem is border detection in ultrasound images. We do so using a new type of active contour. In the process, some important aspects regarding shape representation and modeling, are being addressed. Concretely, we propose

- a curvature-based representation of shape, invariant to similarity transforms (scale, rotation and translation) (Chapter 2, Section 2.5).
- a new algorithm for fast shape alignment without correspondences (Chapter 3, Section 3.4).
- a shape learning method without point correspondences (Chapter 4, Section 4.2).
- a new type of snakes - interpolation snakes - based on cubic splines, which, unlike other snakes
  - have smoothness built-in,
  - allow for spatial hard constraints,
  - incorporate a shape model,

  which makes them suitable for very difficult image domains such as ultrasound. These snakes are defined in Chapter 6 - the centerpiece of the entire thesis.
- a powerful algorithm, for parsing 1D signals as a piecewise constant function, supported mathematical proof (Chapter 7). We apply it to extract feature points for border detection in Chapter 8, Section 8.5.
- an objective, comprehensive and systematic comparison of several active contour algorithms on ultrasound images (Chapter 8).

- a subjective blind study including our interpolation snakes, the traditional snakes [63] and manually traced ground truth boundaries, in order to demonstrate clinical validity for our methods.

The thesis is organized in three parts. The first is dedicated to shape learning (representation and modeling). The second covers border detection using active contours. The last part consists of applications, experiments and results.

A note on the presentation. It seems customary that a literature review be done in the beginning of a thesis, separate from, and followed by the author's research contributions. Since this particular thesis deals with related, but nevertheless distinct topics - representation, modeling and detection of shape - we find it more natural to split the background literature, and seamlessly incorporate the relevant pieces at the appropriate chapters, interleaving the background with the new ideas. This is because we believe it is important to see the entire train of thought. In this sense, the style of this exposition is closer to a book. It goes without saying, of course, that we provide references for everything published elsewhere, so it should be clear when we refer to existing concepts.

This thesis started out as a concrete project of border detection in ultrasound images, at Siemens Corporation Research, in the summer of 2002. The approach was to define and use an active contour with smoothness built-in. Splines come with smoothness built-in, but it was soon apparent that smoothness was not sufficient for a commercial-grade application: the snake would have undefined, uncontrollable behavior when passing through gaps in the boundary of the object (which were not few!). It became apparent that, in order to bring the snake under control at those gaps, we had to introduce shape-prior information in the snake evolution. That is, a shape model was in order. We have explored (and implemented!) a number of registration methods, shape learning techniques, and *many* different "species" of snakes. All the shape representation, shape alignment and model learning methods that we develop in Part 1 of this thesis are a build-up for creating a smooth snake, with knowledge of shape. In the process, we have come up with new algorithms. Sometimes, out of frustration, we looked for distant alternative methods for border detection, suitable for ultrasound. Such is the case, for instance, with the 1D segmentation algorithm in Chapter 7.

Part I

Shape Learning

Chapter 2

Shape Representation

Shape is arguably the most important feature in vision. This part of the thesis is about building mathematical representations of shape and shape models that can be used for object detection and recognition.

We begin this chapter by defining the concept of 2D shape (Section 2.1). We then discuss various mathematical representatations of it, and some of their properties. With the exception of the curvature representation of shape (Section 2.5), the current chapter constitutes mostly foundational material, and is included in the thesis for completeness. The representations detailed here will be used at one point or another in later chapters.

2.1 Shape

The foundation block of building a model for a class of shapes is the problem of curve registration. The crux of the matter is captured in the following very general and apparently simple question: how can we compare two (or more) shapes? The goal is, of course, to tell whether or not the two shapes represent the same object. If the two shapes do represent the same object, they are similar, but perhaps not identical. This is the case, for example, if the images were taken at different times, or from (slightly) different viewing angles (Figure 2.1).

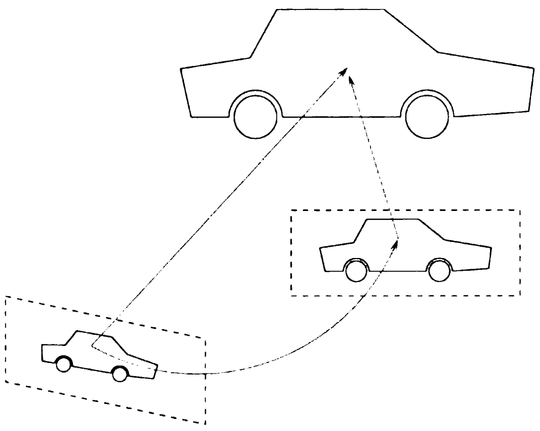


Figure 2.1: Different views of the same object produce similar, but not identical shapes.

Before a similarity measure is built between two shapes, one must understand the nature of the differences between shapes. On one hand, we can capture different contours of the very same object due e.g., to different viewing angles, or due to the motion of the object. Thus, we may have apparently different contours arising from the same objects. These contours will differ e.g., by a rigid or affine transformation, if the object is rigid. Things are more complicated with non-rigid objects (e.g., heart, lungs, etc.), which can deform by non-linear transformations. These differences in the apparent shape of an object are not due to the intrinsic geometry of the object itself, so we term them *extrinsic* differences. It is therefore apparent that when comparing shapes, we must factor out a suitable group of 2D transformations (most commonly rigid or scaling), in order to rule out non-essential differences in shape (the extrinsic differences).

On the other hand, there are *intrinsic* differences between shapes. Intuitively, these are the shape differences that distinguish, say, an apple from a banana, or an airplane from a car. It is the intrinsic differences that a similarity measure between shapes must capture.

Mathematically, then, we compare shapes by means of a similarity metric $d(\cdot,\cdot)$, which is invariant under the appropriate group of transformations (Section 2.1.1). That is, for any two curves γ and δ, the distance between them does not change under the action of arbitrary transforms T and S in the group:

$$d(T(\gamma), S(\delta)) = d(\gamma, \delta)$$

If the metric d is not invariant under the desired group of transformations, then our only chance of comparing curves is to first align them (effectively factoring out the extrinsic differences, usually scale, translation and rotation), and then compute the distance between the aligned curves. Specifically, if γ and δ are the curves, and d is the non-invariant metric, then we seek a transformation T (from the appropriate group of transformations), which minimizes the quantity $d(T(\gamma), \delta)$.

Then, with this T, we can define an invariant metric:

$$d_{inv}(\gamma, \delta) = d(T(\gamma), \delta) \tag{2.1}$$

Finding the transformation T that factors out the extrinsic differences is usually termed *curve registration.*

Various metrics have been used in the literature for shape comparison: the *Hausdorff distance* was used by Huttenlocher *et al.* [54], while Dubuisson and Jain used a modified version of the Hausdorff distance [32]. Ton and Jain [104] use *support functions*; Sclaroff and Pentland [94] use *strain energy.* Least-squares (squared error) and Procrustes metrics are popular among statisticians (Goodall [45], Dryden and Mardia [30]); also used by Horn [52, 53], Besl and McKay [7], Gold *et al.* [43], Cootes [21], and Duta *et al.* [35].

2.1.1 Classes of 2D transformations

The choice of the appropriate group of transformations that describes the extrinsic differences is crucial, and depends on the problem at hand. We describe next some of these classes of transformations most frequently used in computing a model from a set of training samples. Figure 2.2 shows these transforms.

Unless otherwise stated, $T : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ represents a 2D transformation, X is an arbitrary point in $\mathbb{R}^2$ and τ is a translation vector.

Linear transformations are, by definition, transformations that preserve colinearity, i.e., map lines onto lines. These are usually the most appropriate transformations when building a model from a set of training samples, but they can also be used in comparing two standalone shapes.

- **Rigid transformations.** These are transformations that consist of a rotation followed by a translation:

$$T(X) = R_\phi \cdot X + \tau$$

  where R_ϕ is a rotation of angle ϕ. They preserve colinearity, parallelism, lengths, angles and areas.

- **Scaling transformations.** These are also called *similarity* transformations. First, a rotation is performed, followed by an *isotropic* scaling, and last, a translation:

$$T(X) = \rho \cdot R_\phi \cdot X + \tau$$

  where $\rho > 0$ is the scale factor. The rotation and the scaling can be interchanged with each other, but not with the translation. As with the rigid transformations, scaling preserves colinearity, parallelism and angles, but the lengths are multiplied by the scale ρ, and areas differ by a factor of ρ^2.

It is worthwhile to notice that if we identify the real plane $\mathbb{R}^2$ with the complex plane $\mathbb{C}$ in the usual way ($X = (a, b) \leftrightarrow z = a + i \cdot b$), then the scaling transformation can be seen as a map $T : \mathbb{C} \longrightarrow \mathbb{C}$ given by

$$T(z) = \omega \cdot z + \tau$$

with $\omega = \rho \cdot (\cos\phi + i \cdot \sin\phi) = \rho \cdot e^{i\phi}$.

- **Affine transformations.** These transformations are described by a 2×2 matrix A and the translation vector τ, for a total of 6 parameters. The matrix A accounts for rotation, *non-isotropic* scale (2 parameters), and shear.

  $$T(X) = A \cdot X + \tau$$

  The affine transforms still preserve colinearity and parallelism.

- **Projective transformations.** These are linear transformations of the projective plane. By definition, the projective plane of dimension k, is the set of lines in $\mathbb{R}^{k+1}$ going through the origin. Explicitly, a 2D affine transformation is of the form

  $$\begin{aligned} x' &= \frac{ax + by + c}{gx + hx + 1} \\ y' &= \frac{dx + ex + f}{gx + hx + 1} \end{aligned}$$

  where a, b, c, d, e, f, g and h are the 8 parameters that define the projective transformation.

In addition to linear transforms, there is obviously the much larger class of non-linear transforms. In general, the non-linear transforms are not used to compute models, but rather for image registration, under the paradigm of *deformable templates* ([58, 70] and references therein).

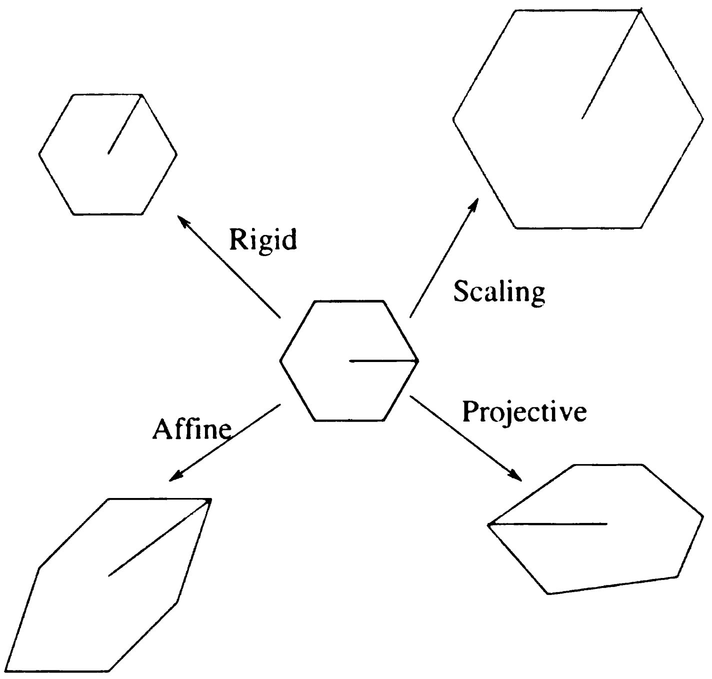


Figure 2.2: Linear transforms. Blue: original shape. Red: linear transformations of the blue shape: rigid (rotation + translation), scaling (preserves shape up to scale), affine (introduces shear and non-isotropic scale, but still preserves parallelism) and projective (only preserves colinearity).

2.2 Explicit representation of shape

Mathematically, a 2D shape is represented as a *parameterized* curve in the plane, i.e., a (smooth) function of one real variable defined on some interval $[a, b]$.

$$\gamma : [a, b] \longrightarrow \mathbb{R}^2, \qquad \gamma(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}.$$

For numerical computations, the curve is sampled at discrete time intervals. This yields an array of points $(\gamma_0, \ldots \gamma_{n-1}) = (x_0, y_0, \ldots, x_{n-1}, y_{n-1})$, which, for convenience, will be denoted by the same symbol γ as the continuous representation. In our computations it will be clear from the context whether we refer to the continuous version or to the discrete version of the curve.

Occasionally, some computations can be carried out in a more concise form in complex notation, by viewing γ as a curve in the complex plane $\mathbb{C}$, via the canonical isomorphism $\mathbb{R}^2 \cong \mathbb{C}$ given by $(x, y) \leftrightarrow z = x + iy$. We shall denote complex curves by $z(t)$, $w(s)$, etc. and use the Greek letters γ, δ, etc. for curves in $\mathbb{R}^2$. With these notations, the discrete version of a (continuous) complex curve $z(t)$ is simply a vector $\mathrm{z} = (z_0, \ldots, z_{n-1}) \in \mathbb{C}^n$.

2.2.1 Parametrization

Let $t = (t_0, \ldots, t_{n-1})$ be the time instants at which γ is sampled, so that $\gamma_i = \gamma(t_i)$. In general, the points γ_i are not equally spaced along γ even if the t_i's are equally spaced. It all depends on the particular parametrization of γ.

Example 2.2.1 *Two parametrizations of the unit circle.*

$$\gamma(t) = \begin{cases} x(t) &= \frac{t^2-1}{t^2+1} \\ y(t) &= \frac{2t}{t^2+1} \end{cases}$$

Sampling from $t = -20$ to $t = 20$ at constant increments of $\Delta t = 0.2$, we get 200 points shown in Figure 2.3 (a), which are obviously non-uniform. On the other hand, if we parameterize the circle by

$$\gamma(s) = (\cos s, \sin s)$$

then we get a nice discretization, in which the points are equally spaced, if we subsample from $s = 0$ to $s = 2\pi$ at constant increments of $\Delta s = 2\pi/200$. See Figure 2.3 (b).

□

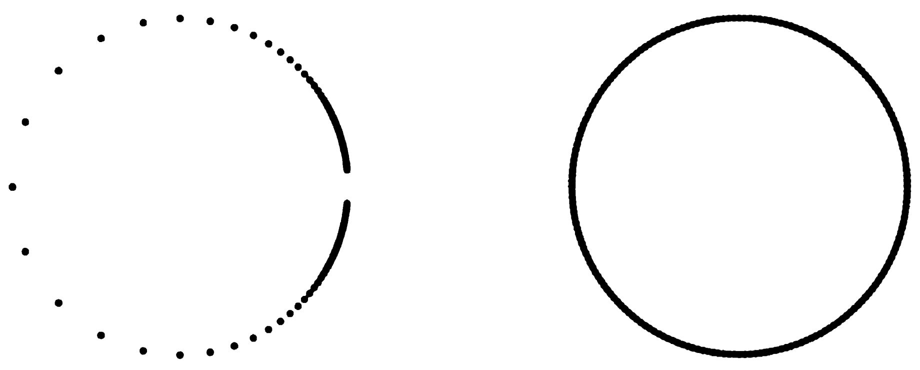


Figure 2.3: Two parametrizations of the unit circle. (a) $x = (t^2-1)/(t^2+1), y = 2t/(t^2+1)$. (b) Re-parametrization by arclength: $x = \cos s$, $y = \sin s$.

Of course, choosing a parametrization with equidistant points is not a matter of aesthetics, but, as it will turn out, it is essential in border detection.

While it is trivial to parametrize the circle in a way that produces a good discretization, things may not be so obvious for arbitrary curves.

2.2.2 Arclength parametrization

Along any smooth curve γ, there are two important vector fields that can be used to describe the local properties of γ: the first and the second derivative. If we imagine a particle moving along the curve, then the first derivative represents the velocity of that particle: it is a vector tangent to γ (at each point) and its magnitude is equal to the speed of the particle. The second derivative, measures the rate of change of the velocity, so it represents the acceleration along the curve. It is still a vector defined at all points on γ, but it is no longer tangent to the curve. We shall use the classical notation γ' and γ'' for the tangent and for the acceleration to γ, respectively. The *length* of γ is defined as the integral of the magnitude[1] of the velocity:

[1]As a matter of notation and terminology, we denote the magnitude of a vector v by $|v|$. We also use interchangeably the terms "length", "norm" and "magnitude".

$$Length(\gamma) = \int_a^b |\gamma'|dt \tag{2.2}$$

Just as the discretization depends on the parametrization, so do the vector fields γ' and γ''. Indeed, if the time parameter t is itself a function of a new parameter $t = t(s)$, then, although $\gamma(t)$ and $\gamma(t(s))$ have the same geometric image, they have different tangent vectors $d\gamma/dt$ and $d\gamma/ds$, related, obviously, by the chain rule:

$$\frac{d\gamma}{ds} = \frac{d\gamma}{dt} \cdot \frac{dt}{ds}$$

Consequently, the computations that involve the tangent and/or the acceleration depend (a priori) on the parametrization. The length of the curve (eq. 2.2) is one exception.

Fortunately, there exists a special parametrization - a "canonical" parametrization, in some sense - that has many desirable properties and which facilitates the computations. Let $\gamma = \gamma(t)$ be any parametrization. One defines the *arclength* parameter on γ as the distance along γ traveled up to time t:

$$s = \int_a^t |\gamma'|$$

A priori, $s = s(t)$ is a function of the original parametrization t. Obviously, s is differentiable and $s'(t) = |\gamma'(t)| \geq 0$. If the velocity γ' is nowhere zero, then s is strictly increasing, and one can compute the inverse of the function $s = s(t)$ and express t as a function of s: $t = t(s)$. We obtain this way a new parametrized curve $\tilde{\gamma}(s) = \gamma(t(s))$ with the following properties:

1. $\tilde{\gamma}$ and γ have the same geometric image.

2. By the chain rule, the velocity $\tilde{\gamma}'$ along the re-parameterized curve has norm 1:

$$|\tilde{\gamma}'| = 1 \tag{2.3}$$

3. The *acceleration* $\tilde{\gamma}''$ is perpendicular to the velocity:

$$\tilde{\gamma}'' \perp \tilde{\gamma}' \tag{2.4}$$

   As $\tilde{\gamma}$ changes convexity, the normal vector $\tilde{\gamma}''$ changes sides, always pointing towards the "interior" of the turn.

4. Last, but not least, the discretization of a curve parameterized by arclength is equally spaced, because the arclength parameter s represents distance *on the curve itself.* Indeed, if the curve is discretized at $s_0 \ldots s_{n-1}$ with $\Delta s = s_{i+1} - s_i = const$ and $\gamma_i = \gamma(s_i)$, then

$$|\gamma_i \gamma_{i+1}| \approx |s_i - s_{i+1}| = \Delta s = const$$

Remark 2.2.1 *A word on notation: because of the fact that γ and $\tilde{\gamma}$ have the same geometric image, it is generally accepted to drop the $\tilde{}$ symbol, and use the same letter γ for both curves, indicating explicitly whether one is working with the arclength parameter ($\gamma = \gamma(s)$) or with the time parameter ($\gamma = \gamma(t)$). Differentiation with respect to the time parameter is indicated using the $'$ ("prime") symbol, whereas derivatives with respect to the arclength parameter s, are indicated using the $\dot{}$ ("dot") symbol. Thus, γ' and γ'' are the first and second derivatives along the curve, in an arbitrary parametrization, while $\dot{\gamma}$ and $\ddot{\gamma}$ are the first and second derivatives with respect to the arclength s.*

In Example 2.2.1, without getting into details, the $(\cos s, \sin s)$ parametrization is in fact the arclength parametrization of the circle. This is the reason the points are equally spaced in Figure 2.3 (b).

Under the arclength parametrization, integrals along the curve are approximated numerically as finite sums:

$$\int_{\gamma} f(\gamma)ds \approx \sum_{i} f(\gamma_i)$$

2.2.3 Curvature

Let γ be a curve parameterized by arclength, and $\dot{\gamma}$ its tangent. Let ν be the unit to γ, chosen so that the frame $(\dot{\gamma}, \nu)$ has positive orientation. Clearly, since $\ddot{\gamma} \perp \dot{\gamma}$, then at each point on γ, the acceleration $\ddot{\gamma}$ is a scalar multiple of the normal ν:

$$\ddot{\gamma} = k_{\gamma} \cdot \nu \tag{2.5}$$

Definition 2.2.1 *The curvature of a curve γ is the scalar k_{γ} in* (2.5).

Trivially, by taking the dot product against ν, we get from (2.5) that

$$k_{\gamma} = \langle \ddot{\gamma}, \nu \rangle \tag{2.6}$$

and

$$|k_{\gamma}| = |\ddot{\gamma}| \tag{2.7}$$

The normal ν does not change from one side to another, as γ changes concavity, but the acceleration $\ddot{\gamma}$ does. Consequently, the sign of the curvature k_{γ} changes as well, the same way $\ddot{\gamma}$ changes.

The next proposition summarizes some properties that we shall use in this thesis:

Proposition 2.2.1 *1. Under an arbitrary parameter t,*

$$k_\gamma = \frac{x' \cdot y'' - x'' \cdot y'}{(x'^2 + y'^2)^{3/2}} \tag{2.8}$$

2. If γ is given implicitly (Section 2.4), as the zero level curve of a surface $f(x, y) = 0$, then the curvature can be computed by

$$k_\gamma = \frac{|f_{xx} f_y^2 + f_{yy} f_x^2 - 2 f_{xy} f_x f_y|}{|\nabla f|^3} \tag{2.9}$$

3. If $a\gamma$ is a scaled version of γ, and s is the arclength on γ, then $s \cdot a$ is the arclength on $a\gamma$ and

$$length(a\gamma) = a \cdot length(\gamma) \tag{2.10}$$

$$k_{a\gamma}(sa) = \frac{k_\gamma(s)}{a} \tag{2.11}$$

2.3 Cubic spline approximation.

Although the presentation of the material in this thesis is top-down (i.e., first the theory and then the applications), the new concepts and algorithms were derived in a bottom-up manner, starting from a concrete practical problem: border detection in ultrasound images (Section 8.2). In general, serious algorithms in medical imaging require clinical validity, and that, in turn, requires that the automatically detected boundaries be compared against manually traced boundaries by a human medical expert for training and evaluation purposes. Invariably, the expert does not draw the entire contour as a continuous curve, as one might imagine, but rather clicks on some 20-30 points on the contour. The traced contours will most likely have to

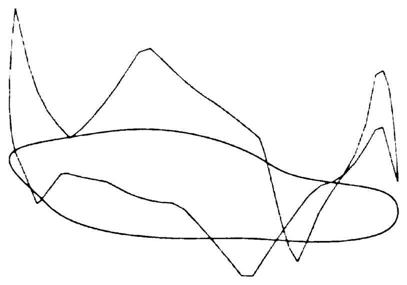


Figure 2.4: 3D view of a 2D contour (blue) with its curvature (green) drawn on top of the contour.

be matched across patients, so if the organ being traced happens to have distinct anatomical landmarks (usually rigid tissue like bones, teeth, vertebrae but also some brain regions), then the human expert will mark those, in equal number in each image. This way, *point correspondences* are established *a priori*, i.e., before any preprocessing and shape learning, in the most natural way. If, on the other hand, there are no, or very few, clearly defined anatomical landmarks (e.g., kidney, bladder), then there is no choice but to mark more or less arbitrary points on the contour, in different numbers across patients, leaving it to the computer to recover or establish the point correspondences. Under this inherent non-uniformity we have determined that the best way to represent the hand-drawn contours is to interpolate the markers using a cubic spline, parameterized by arclength, using a judicious choice of the spline parameters. See Figures 2.5, 2.6 and 2.7. It is central to our applications, so we provide all the necessary details next.

Following [28] (Chapter IV) and [29] an *interpolation spline* is defined in 2D by

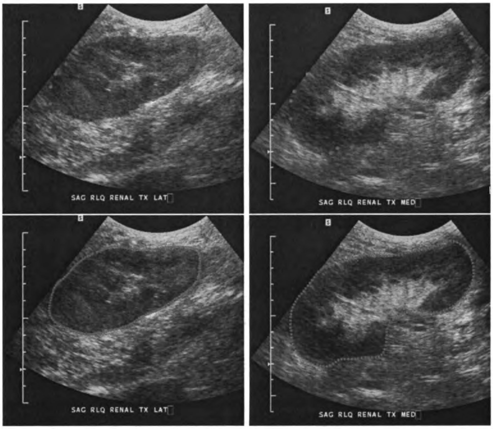


Figure 2.5: Contours marked by a human expert on kidney ultrasound. Top row: markers (two different patients). Observe the sparseness, lack of anatomical landmarks, non-uniformity and the different number of markers across patients. Bottom row: corresponding kidney contour represented as a dense, cubic interpolation spline determined by the markers, parameterized by arclength (equally spaced points). Both splines are discretized using the same number of points (150).

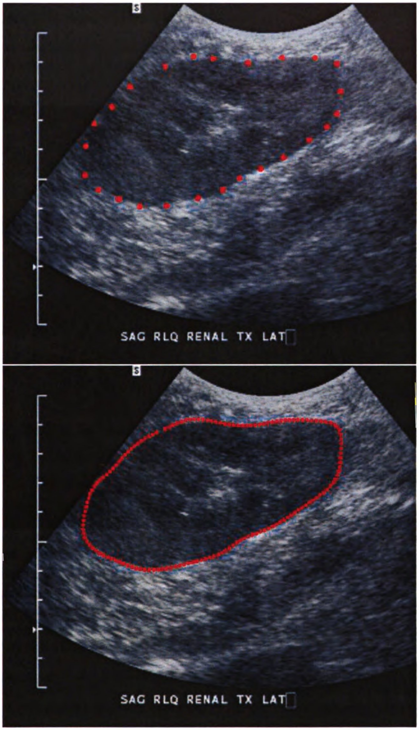


Figure 2.6: Detail from Figure 2.5: patient one (left column).

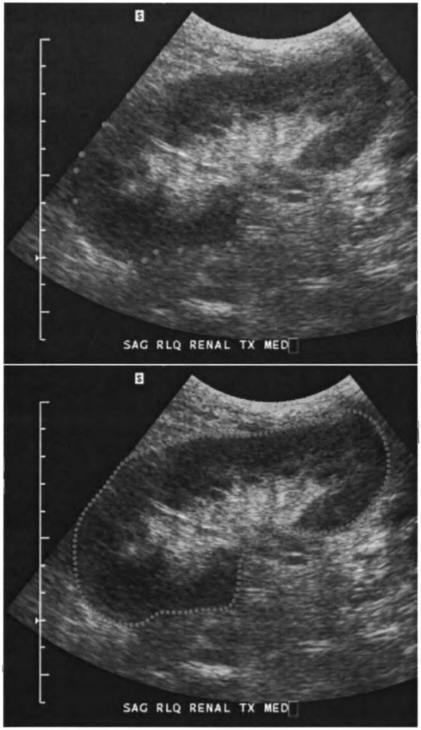


Figure 2.7: Detail from Figure 2.5: patient two (right column).

the following elements:

- **break points:** a strictly increasing set of real numbers $a = \xi_0 < \xi_1 \cdots < \xi_l = b$
- **control points:** a set of data vectors: $y_0, \ldots y_l \in \mathbb{R}^2$ one y_i for each ξ_i.
- **degree:** a positive integer k.

Given this data, the *interpolation spline of degree k*, is defined as a curve

$$\gamma : [a, b] \longrightarrow \mathbb{R}^2$$
$$t \longrightarrow \gamma(t) = (x(t), y(t))$$

with the following properties:

- γ passes through the control points y_i at ξ_i, that is

$$\gamma(\xi_i) = y_i \quad \forall i$$

- componentwise (i.e., each of x and y), γ is a degree-k polynomial on each interval $[\xi_i, \xi_{i+1})$
- the pieces fit smoothly at the break points: γ is of class C^{k-1}, i.e., it has $k - 1$ continuous derivatives.

Of particular importance are the *cubic* interpolation splines ($k = 3$), due to the so called **smoothest interpolation** property ([28] Corollary 7, Chapter V):

Proposition 2.3.1 *Given breaks $a = \xi_0 < \cdots < \xi_l = b$ and corresponding control points $y_0, \ldots, y_l$, consider the following minimization problem with constraints:*

$$E(\gamma) = \int_a^b |\gamma''(t)|^2 dt \tag{2.12}$$

$$\gamma(\xi_i) = y_i \quad \forall i$$

Then, over the space $\mathbf{C}^2$ *of twice continuously differentiable functions on* $[a, b]$*, the minimum (a) exists, (b) is unique, and (c) is achieved by the (necessarily unique)* **cubic interpolation spline** *on the given breaks and control points.*

Minimization of the total second derivative (as opposed to other derivatives), is important because of its geometric significance: for *any* curve (i.e., not just splines), $|\gamma''|$ measures the amount of bending of the curve. For this reason, the energy (2.12) is called *bending energy*. In fact, under the canonical parametrization on γ, the *curvature* of γ is defined precisely as

$$k_\gamma = |\ddot{\gamma}| \tag{2.13}$$

where, following the convention in Remark 2.2.1, $\ddot{\gamma}$ is the second derivative of γ with respect to the arclength parameter s.

The computation of a cubic interpolation spline can be carried out in *linear* time in the number l of control points. That is, given $t \in [\xi_i, \xi_{i+1}]$, the complexity of computing $\gamma(t)$ is $O(l)$, (where l is the number of control points), because, as we shall see, the first derivatives at all break points must be determined. Since these derivatives do not depend on t (but only on the break points), they can be computed once and for all, making computation of $\gamma(t)$ an $O(1)$ process. We present in detail the numerics of cubic interpolation splines in Appendix B. In the end, we end up with a function

$$\gamma : [a, b] \longrightarrow \mathbb{R}^2$$

$$t \longrightarrow \gamma(t) = (x(t), y(t))$$

$$x(t) = \begin{cases} P_1(t) \; if \; \xi_0 \le t < \xi_1 \\ \dots \\ P_{l-1}(t) \; if \; \xi_{l-2} \le t < \xi_{l-1} \end{cases} \qquad y(t) = \begin{cases} Q_1(t) \; if \; \xi_0 \le t < \xi_1 \\ \dots \\ Q_{l-1}(t) \; if \; \xi_{l-2} \le t < \xi_{l-1} \end{cases} \tag{2.14}$$

where P_i and Q_i are cubic polynomials defining the x and the y components, respectively, on the interval $[\xi_{i-1}, \xi_i)$.

Numerically, the spline is discretized by applying Equations (2.14) on an array $t = (t_0, \dots, t_{n-1})$, where the number n is the user-defined number of points on the spline. Most commonly, the t_i's are equally spaced, i.e., the spline is generated by increasing uniformly the time parameter. The resulting points $\gamma(t_i)$ on the spline, are, of course, not uniformly spaced, as t is not the arclength parameter (Figure 2.8, left). We explain how this can be corrected, next.

2.3.1 Arclength parametrization of an interpolation spline

We can compute an (approximately) arclength parameterized spline, by manipulating the break points ξ_i. If y_i are the user-defined control points, then we choose $\xi_0 = 0$, and $\xi_i - \xi_{i-1} = |y_i y_{i-1}|$ for $i > 0$, so that the length *along the curve* between two consecutive control points y_{i-1} and y_i is approximated (to the extent that the *chord* $y_{i-1}y_i$ approximates the corresponding *arch* along the spline) by $\Delta\xi = \xi_i - \xi_{i-1}$. By choosing the breaks this way, the usual (uniform) parameter t along the real axis, becomes - approximately - the arclength parameter along the resulting spline $\gamma = \gamma(t)$.

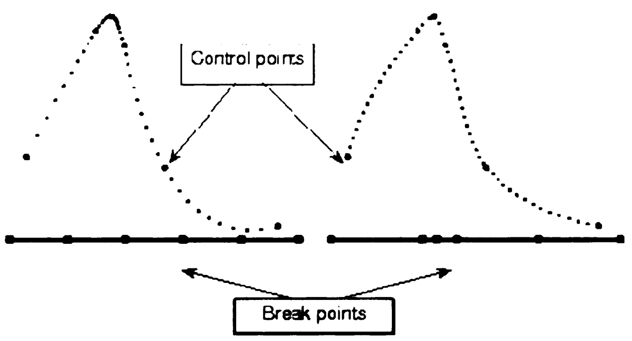


Figure 2.8: Cubic interpolation spline (blue). Red: control points. Black: break points. The spline on the left has equidistant breaks, and is not parametrized by arclength (the density is higher near points of high curvature). The spline on the right is built on the same control points as the one on the left. Choosing the breaks ξ_i by $\xi_i = \xi_{i-1} + |y_{i-1}y_i|$ produces an arclength parameterized spline.

Remark 2.3.1 *This choice of the breakpoints, which seamlessly produces a curve parametrized approximately by arclength, is not possible with types of splines that do not pass through their control points (e.g., B-splines, Bezier); it is only possible with interpolation splines, because of the particular significance of the control parameters, as points on the curve.*

2.4 Implicit representation

In addition to the explicit representation of shape as a parametric curve, there exists a qualitatively different representation. If $f(x, y)$ is a function of two independent variables, then the equation $f(x, y) = const$ defines y *implicitly*, as a function of x, provided that f is smooth and $\partial_y f$ is nowhere 0. Each constant defines a different curve, so f determines, in fact, a family of curves, called the *level sets* of f (Figure 2.9).

We digress here, to explain why the level-set representation of a curve is useful.

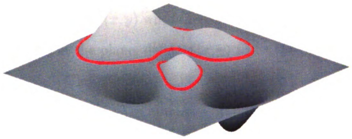


Figure 2.9: Implicit (level sets) representation of curves. Given a smooth function $z = f(x, y)$, each constant c defines implicitly a curve $c = f(x, y)$.

The traditional snakes, as defined by Kass, Witkin and Terzopoulos [63], have several drawbacks, many of which have been addressed in subsequent studies [2, 20, 42, 111], but two of which are that (a) the algorithms are numerically unstable, and (b) the snake cannot change the topology (e.g., if the snake is initialized as a closed simple curve, then it cannot break to converge to a configuration with several connected components).

A *level-set snake* is the zero level set of a moving surface $z = f(x, y)$. Clearly, if the surface has more than one hump, the snake will change its topology (Figure 2.9). The problem, of course, is to control the movement of the surface, so that its zero level set converges to the salient features in an image. We consider this revolutionary approach to curve evolution so important that we describe it in some detail in Section 5.2.

2.4.1 Distance map representation of shape

We have seen that, cutting a surface $z = f(x, y)$ with a plane $z = const$ produces a curve. Conversely, given a parametric representation of a plane curve $\gamma(t)$, it is possible to find a function $z = f(x, y)$ whose zero level set is precisely γ. Specifically, for any point $X = (x, y)$ in the plane, define $f(x, y)$ to be the distance from X to γ:

$$f(X) = \min_t |X - \gamma(t)|. \tag{2.15}$$

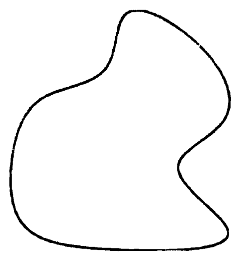
Clearly, a point X is at distance 0 from γ if and only if X is on γ, so γ is precisely the zero level set of f. The function f is called the distance map representation (or the *chamfer transform*) of γ [4, 9] (Figure 2.10).

If the curve γ is simple and closed, then γ is usually represented by the *signed* distance map, which is the same as the usual distance map, except that the points in the exterior have negative sign (but the same magnitude as before).

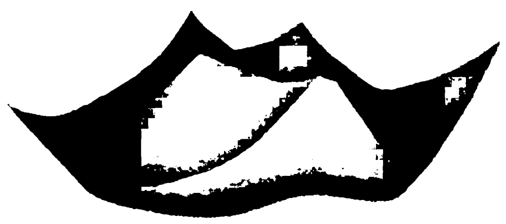
Remark 2.4.1 *A word on the sign convention. If ϕ is any surface, then the gradient $\nabla\phi$ is orthogonal to the level sets, and points in the direction in which ϕ increases most rapidly. As such, if γ is the zero-level set of ϕ and if γ is a simple, closed curve, then the vector $\nu := \nabla\phi/|\nabla\phi|$ is the* inward *unit normal to γ if $\phi < 0$ outside γ and $\phi > 0$ inside. With this sign convention, the unit circle, with the positive orientation (counterclockwise) has positive curvature (as it should).*

Regarding the complexity of computing the distance map, if I is an $m \times n$ image, and γ is a curve of length L in the image, then a straightforward, exact computation of the distance map takes $O(m \cdot n \cdot L)$ because for each image pixel one must compute the distance to each curve pixel. If L is of the order of a few hundred points, then the algorithm is too slow.

Borgefors [9, 10] introduced an algorithm which requires only two passes through the image (hence with complexity $O(m \cdot n)$, which computes a good approximation



(a)



(b)

(c)

Figure 2.10: Distance map representation of a plane curve. (a) Explicit representation as a (discrete) parametrized curve. (b) Distance map representation. (c) Signed distance map representation.

of the distance map. Her work was directed toward image matching using the distance map (termed *chamfer matching*). Chamfer matching pre-dates Borgefors, being introduced in 1977 by Barrow *et al.* [4].

Yet another way of computing the distance map of a given curve, is to have a moving front evolve along its own normal, with constant speed of 1. This method is most readily implemented using level-sets, and, naturally, it is the method of choice in the level-set community.

2.5 Curvature-based representation of shape

The metrics we have encountered so far were not invariant to plane transformations (Section 2.1.1), for the very reason that - implicitly or explicitly - the curves were represented in terms of their cartesian coordinates (x, y). This gives rise to the entire problem of registration (Chapter 3): bring the curves together, to some standard pose, and then compare them.

We develop in this section a measure of similarity between curves which is *invariant* to scale, rotations, and translations, i.e., invariant to similarity transforms.

To show the intuition behind this new representation, we reason in the discrete case first. Let, as before, $\gamma = (\gamma_0, \dots, \gamma_{n-1})$ be a curve. Assume γ is parameterized by arclength, which translates mathematically that the length between any two consecutive points on γ is 1:

$$|\gamma_i \gamma_{i+1}| = 1$$

Since all the lengths of the constituent segments of γ are known, then, all it takes to move along γ is to know how much to steer at each of the γ_is. See Figure 2.11.

This suggests a representation of a curve using the angles between the *vectors* $\gamma_{i-1}\gamma_i$ and $\gamma_i\gamma_{i+1}$, adjacent to γ_i. Let $\Delta\theta_i = \theta_i - \theta_{i-1}$ be this angle. As shown

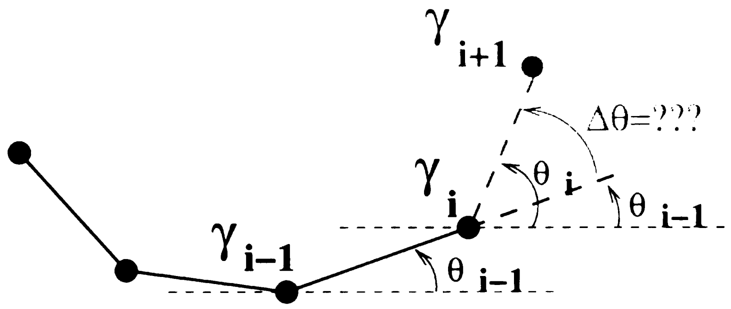


Figure 2.11: Moving along a discrete, equally spaced curve γ only requires to know the angle between consecutive segments.

in Figure 2.11, for each i, θ_i is the angle between the x-axis and the vector $\gamma_i\gamma_{i+1}$. All these angles can obviously be computed from the cartesian representation of γ. Conversely, knowing an array $\Delta\theta = (\Delta\theta_1, \ldots, \Delta\theta_{n-1})$ permits the recovery of the cartesian representation of γ, modulo a translation and a rotation (uniquely determined once a starting point γ_0 and an initial direction θ_0 are specified).

Intuitively, the "amount of steering", must have something to do with curvature. It is indeed the case, and we shall explain this in the continuous setting. Let now $\gamma : [0, L] \longrightarrow \mathbb{R}^2$ be a smooth curve (e.g., of class $\mathbf{C}^2$). At an arbitrary point $\gamma(s)$, let $\theta(s)$ be the angle between the tangent $\dot{\gamma}$ and the x-axis, that is

$$\dot{\gamma} = (\cos\theta, \sin\theta) \tag{2.16}$$

Proposition 2.5.1 *If k is the curvature of γ, then*

$$k = \dot{\theta} \tag{2.17}$$

Proof: Take the derivative in (2.16), then the dot product against the normal $\nu = \dot{\gamma}^{\perp} = (-\sin\theta, \cos\theta)$. Apply Equation (2.5).

□

From this proposition, it is clear that the curve γ is determined by its curvature alone, modulo translations and rotations. Indeed, knowing k, then by (2.17) and (2.16) we have

$$\begin{aligned}\theta(s) &= \theta_0 + \int_0^s k(\sigma)d\sigma \\ x(s) &= x_0 + \int_0^s \cos\theta(\sigma)d\sigma \\ y(s) &= y_0 + \int_0^s \sin\theta(\sigma)d\sigma\end{aligned} \tag{2.18}$$

Remark 2.5.1 *A representation of a curve γ in terms of its curvature k would be invariant to translations (as k is defined in terms of derivatives), and to rotations (as they preserve dot products). However, the curvature is inversely proportional to scale. For instance, the curvature of a circle of radius r is $1/r$. The larger the circle, the smaller the curvature.*

All these facts are well known to geometers. What we bring new to computer vision here is an invariant metric between curves, based on curvature. To this end, let γ and δ be two curves parameterized by arclength, and let L_γ and L_δ be their respective lengths, and k_γ and k_δ their respective curvatures. We would like to define the similarity between γ and δ by comparing their curvatures, e.g., by their L^2 distance:

$$\int |k_\gamma - k_\delta|^2$$

Of course, since k_γ is defined on $[0, L_\gamma]$ and k_δ is defined on $[0, L_\delta]$, this integral does not make sense unless the curves have the same length. This problem is solved if we first normalize γ and δ to have length 1. For any arclength parameterized curve $\alpha : [0, L] \longrightarrow \mathbb{R}^2$ we define

$$\begin{aligned}\alpha_1 &: [0,1] \longrightarrow \mathbb{R}^2 \\ \alpha_1(t) &= \frac{\alpha(tL)}{L}\end{aligned} \tag{2.19}$$

The next lemma expresses the derivatives, arclength, length, and curvature of α_1 in terms of those of α.

Lemma 2.5.1 *If α and α_1 are as above, then:*

1.

$$\frac{d\alpha_1}{dt}(t) = \dot{\alpha}(tL), \qquad \frac{d^2\alpha_1}{dt^2}(t) = L \cdot \ddot{\alpha}(tL) \tag{2.20}$$

2. if s is the arclength on α, then $t = s/L$ is the arclength on α_1. Also, $L_{\alpha_1} = 1$.

3. $k_{\alpha_1}(t) = L \cdot k_\alpha(tL)$.

Proof: Trivial. □

Using the length-normalized curves, we can make the following

Definition 2.5.1 (Curvature distance)

$$\begin{aligned} d(\gamma,\delta) &:= \int_0^1 \left| k_{\gamma_1}(t) - k_{\delta_1}(t) \right|^2 dt \\ &= \int_0^1 \left| L_\gamma \cdot k_\gamma(tL_\gamma) - L_\delta \cdot k_\delta(tL_\delta) \right|^2 dt \end{aligned} \tag{2.21}$$

Since curvature is invariant under rotations and translations, so is d. Moreover, due to the unit length normalization of the curves, d is also scale-invariant. Clearly d is *symmetric*, $d \geq 0$ (*positive*), and also satisfies the *triangle inequality*. However, precisely because of its invariance, it is *not* positive-definite. This means that $d(\gamma,\delta) = 0$ does not imply $\gamma = \delta$ (this latter equality does not even makes sense, as γ and δ have a priori different lengths). For instance, scale invariance means that $d(\gamma, a \cdot \gamma) = 0$, i.e., this distance does not distinguish between a curve and any scaled version of it. The following proposition shows what happens when $d = 0$.

Proposition 2.5.2 $d(\gamma, \delta) = 0$ *if and only if* $\gamma_1(t) = \delta_1(t)$, $\forall t \in [0, 1]$.

That is, the curvature metric (2.21) *is a positive-definite metric on the space of curves of unit length.*

Proof: Since by definition $d(\gamma, \delta) = d(\gamma_1, \delta_1)$, we have to prove the equivalence for curves of length 1. Assume therefore that $L_\gamma = L_\delta = 1$. Of course, $d(\gamma, \delta) = 0$ means the curves have the same curvature. The proof follows now from (2.18)

□

There is an additional complication in the case of *closed* curves, arising from the possibility of a phase shift. We want the metric d to be invariant to shifts in the arclength parameter along the curve. That is, if γ is a closed curve, and $\delta(s) = \gamma(s+a)$ is a phase-shifted copy of γ, we want $d(\gamma, \delta) = 0$. This requirement is natural, since γ and δ have the same geometric image. However, their curvatures are also shifted by a:

$$k_\delta(s) = k_\gamma(s + a)$$

so, if we let L be the length (of both γ and δ)

$$\begin{aligned} 0 &= d(\gamma, \delta) \\ &= \int_0^1 |k_\gamma(tL) - k_\delta(tL)|^2 dt \\ &= \int_0^1 |k_\gamma(tL) - k_\gamma(tL + a)|^2 dt \end{aligned}$$

which implies

$$k_\gamma(s) = k_\gamma(s+a) \qquad \forall s \in [0, L]$$

That is, a is a period of the curvature (of course, both γ and its curvature k_γ are periodic, and L is a period because γ is closed, of length L, but a is a smaller period than L). Let $T > 0$ be the smallest period of k_γ. Then, necessarily $a = nT$ for some integer n. This is a necessary condition for the distance to be 0 on phase-shifted curves. Most commonly though, this condition is not satisfied [2], so curves with identical geometric image will have non-zero distance due to shift in the parameter along the curve.

The remedy is, of course, to apply all possible phase shifts to *one* of the curves, and take the minimum.

Definition 2.5.2 (Curvature distance for closed curves)

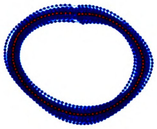
$$\begin{aligned} d(\gamma, \delta) &:= \min_{a \in [0,1]} d(\gamma_1, \delta_1(\cdot + a)) \\ &= \min_{a \in [0,1]} \int_0^1 \left| k_{\gamma_1}(t) - k_{\delta_1}(t+a) \right|^2 dt \end{aligned}$$

The quantity defined by Equation (2.22) measures shape similarity between two closed curves, and is *invariant* to similarity transforms and to phase shifts.

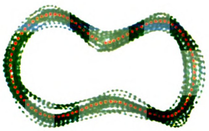
Example 2.5.1 *We consider in this example two classes of shapes: a circular shape, and a "bow tie" shape. Each of the two classes consists of* 30 *synthetically generated samples, drawn from a gaussian distribution around their respective mean. The mean and the samples were defined as arclength-parameterized, cubic interpolation splines. We then computed the* **intra-class** *($d(\gamma_i, \gamma_j)$ for all γ_i, γ_j within the same class, and*

[2]One example when we can shift the parameter s on a curve by a multiple of the period T is if γ resembles a circular saw blade.

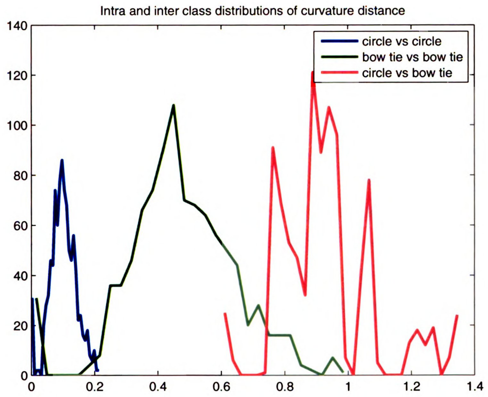
inter-class *distances ($d(\gamma_i, \delta_j)$ with γ_i and δ_j in different classes). We obtain this way three distributions of distances, shown in Figure 2.12 (c). Curves from within the same class tend to be closer to each other, than curves from different classes.*



(a)



(b)



(c)

Figure 2.12: Two classes of shapes, and the distributions of *inter* and *intra* class curvature distances. (a) Circular class: 30 samples (blue) normally distributed around a user-defined mean (red). (b) "Bow-tie" class: 30 samples (green) normally distributed around a user-defined mean (red). (c) Distributions of curvature distances $d(\gamma_i, \delta_j)$. Blue: both γ_i and δ_j are from the circular class. Green: both γ_i and δ_j are from the bow-tie class. Red: γ_i is circular, and δ_j is a bow-tie shape. Note the separability of each of the within-class distributions (the blue and red distributions) from the intra class distribution of distances (green).

Chapter 3

Shape Alignment

Curve alignment (or registration) and shape similarity are two ingredients towards building a shape model. We define the problem of registration in Section 3.1 and provide explicit formulas for the transform that aligns two curves, in the case of similarity, affine and thin-plate-spline transforms (TPS), under the assumption of known point correspondences (Section 3.2). We review some of the representative existing registration algorithms in the case of unknown point correspondences (Section 3.2.2), and discuss as a special case distance map registration (Section 3.3). In the last section, we propose a fast and practical registration algorithm still in the case of unknown correspondences, which works best for elongated contours in 2D.

3.1 Curve registration

Suppose now that we have two curves γ and δ which we want to register (for instance for the purpose of defining an invariant metric on the curves, i.e., a shape distance). That is, we want to find a transformation $T : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ (e.g., from one of the groups of linear transformations in Section 2.1.1), such that

$$T(\gamma) = \delta. \tag{3.1}$$

Clearly, there is little chance that Equation (3.1) has an exact solution in T. For one, in the discrete setting, (3.1) entails that γ and δ have the same number of points, which may not be the case. The next best thing then, is to solve this equation approximately, by specifying an optimality criterion $E = E_{\gamma,\delta}(T)$ and by finding the transform T optimal with respect to E:

$$T = \arg\min E_{\gamma,\delta}(T) \tag{3.2}$$

Computation of the optimality metric E may, or may not require known point correspondences between the curves to be registered (Section 3.2). Most notably, the most straightforward metric between two parametric curves - the L^2 norm - involves pointwise subtraction of the two curves, so one must know, for each point on one curve, what its counterpart on the other curve is.

On the other hand, there are metrics that do not require known point correspondences. These are metrics that compare *each* point on one curve, against *all* points on the other curve. We present some of these below.

To begin with, the distance between a point x and a set Y (in any Euclidean space) can be naturally defined as:

$$d(x, Y) = \inf_{y \in Y} |x - y| \tag{3.3}$$

Next, the distance between two sets X and Y (in our case these will be curves in the plane) can be defined as the largest distance from any point in X, to Y:

$$d(X, Y) = \sup_{x \in X} d(x, Y) \tag{3.4}$$

This notion of distance between two sets (Equation (3.4)) was introduced for the very first time in 1905 by Dimitrie Pompeiu - a distinguished Romanian mathematician [87]. It was originally termed the *écart* [1] of X and Y, and it is sometimes referred to today as the *directed Hausdorff distance* between X and Y. The distance between a point and a set (3.3) is known today in the computer vision community as the *chamfer distance (transform)*. Certainly, the distance (3.4) is not symmetric. In the same work [87], Pompeiu defined the *mutual distance (l'écart mutuel)* of the two sets as the sum

$$D(X, Y) = d(X, Y) + d(Y, X) \tag{3.5}$$

The Pompeiu distance (3.5) was modified by Felix Hausdorff in his classical 1914 book *Grundzüge der Mengenlehre* [2] [46] (republished and translated from German multiple times; for an English translation of the 3-rd edition, see for instance [47]). Hausdorff defined what is now the very popular *Hausdorff distance* by replacing the sum in (3.5) by the supremum:

$$H(X,Y) = \sup\{d(X,Y), d(Y,X)\} \tag{3.6}$$

The supremum in Equation (3.4) makes the *écart* distance sensitive to outliers. This can be alleviated by replacing the supremum with the average [32]:

$$d_{av}(X,Y) = \frac{1}{|X|}\int_X d(x,Y)dx \tag{3.7}$$

Accordingly, the *average Hausdorff distance* can now be defined in a similar way to the regular Hausdorff distance (3.6):

$$H_{av}(X,Y) = \sup\{d_{av}(X,Y), d_{av}(Y,X)\} \tag{3.8}$$

We shall use the term *Hausdorff-type* metrics for any of the distances (3.4), (3.5),

[1] pit, chasm.

[2] Basics of Set Theory.

(3.6), (3.7) and (3.8). If $D(X,Y)$ is any of them, then an optimality criterion for registering two curves γ and δ can be defined by

$$E_{\gamma,\delta}(T) = D(T(\gamma),\delta) \tag{3.9}$$

One issue with Hausdorff-type metrics is the complexity. For discrete curves of about the same size, the complexity is quadratic in the number of points. This complexity is prohibitive for an iterative minimization of (3.9), but there are ways to reduce the complexity to linear (at some cost) discussed in some detail in Section 3.3.

We present some known registration techniques in Sections 3.2 and 3.3.

3.2 Registration of parameterized curves

Mathematically, if $\gamma(s)$ and $\delta(t)$ are two curves, with a priori different parameters s and t, then the *point correspondence problem* consists of finding a re-parametrization $s = h(t)$ *in some optimal way*, i.e., establishing a meaningful correspondence between the points of the two curves. There are several proposed solutions for finding the correspondences (Section 3.2.2), each with its strengths and weaknesses, and with its own optimality metric, but there is no dominant method, nor a demonstrable best method over the others.

Only after introducing the re-parametrization h does it make sense to define the squared error energy:

$$E(T,h) = \frac{1}{2}\int |T(\gamma(h(t))) - \delta(t)|^2 dt. \tag{3.10}$$

The discrete version of this is

$$E(T,h) = \frac{1}{2}\sum_i |T(\gamma_{h(i)}) - \delta_i|^2. \tag{3.11}$$

An invariant similarity measure[3] between the two curves γ and δ can now be defined by

$$d^2(\gamma, \delta) = \min_{T,h} E(T, h). \tag{3.12}$$

3.2.1 Known correspondences

Landmark registration is one of the simplest cases of registration of parametric curves, because it takes place under the following assumptions:

- All curves to be registered have the same number of points.
- The point correspondences are known: all curves are labeled so that for each index i, the i^{th} point on one curve is homologous with the i^{th} point on any other curve, that is $h(i) = i$.

The squared error energy (Equation (3.11)) only depends on T now. In the case of scaling and affine transforms, it is possible to find the parameters of the minimizing T algebraically (Propositions 3.2.1 and 3.2.2 below).

Scaling registration and the Procrustes distance

This problem was first solved by Horn [52, 53] for points in 3-D, and then revived by other authors (e.g., Cootes [21], Dryden and Mardia [30]) in the context of plane curve registration and model learning. We present here a 2-D version using complex numbers, which is one instance when working over the field $\mathbb{C}$ of complex numbers simplifies computations.

Let $z = (z_1, \dots, z_n)$ and $w = (w_1, \dots, w_n)$ be two (discrete, complex) curves with the same number of points. These are vectors in $\mathbb{C}^n$ so the usual L^2 norm in $\mathbb{C}^n$ can be used to define a distance between the shapes z and w which *non-invariant* under scaling transform (so *a fortiori* non-invariant under more general transforms):

[3]This measure is not a metric *per se*, because it lacks symmetry, but it can be symmetrized.

$$d(z, w) = |z - w|^2 \tag{3.13}$$

Equation (2.1) and the arguments leading to it show how one can turn E into an invariant metric between curves, i.e., into a distance on *shapes*. We have seen in Section 2.1.1 that a similarity transformation is of the form

$$T(\omega) = \lambda \cdot \omega + \tau \qquad \forall \omega \in \mathbb{C}$$

for some unknown complex parameters λ and τ. We must find these parameters by minimizing the squared error energy:

$$\begin{aligned} E_{z,w}(\lambda, \tau) &= \sum_j |T(z_j) - w_j|^2 \\ &= \sum_j (\lambda z_j + \tau - w_j) \cdot \overline{(\lambda z_j + \tau - w_j)}. \end{aligned} \tag{3.14}$$

Proposition 3.2.1 *If $T(z) = \lambda z + \tau$ minimizes the squared error energy (Equation 3.14), then its parameters are given by*

$$\lambda = \frac{\langle w', z' \rangle}{|z'|^2} \tag{3.15}$$

$$\tau = -\lambda z_0 + w_0 \tag{3.16}$$

where $\langle \cdot, \cdot \rangle$ denotes the complex inner product on $\mathbb{C}^n$, z_0 and w_0 are the centroids of z and w, respectively, and $z' = (z_1 - z_0, \dots z_n - z_0)$ and $w' = (w_1 - w_0, \dots w_n - w_0)$ are the zero-mean curves.

The minimum value of the energy is

$$E^{\min}_{z,w} = |w'|^2 - \frac{|\langle w', z' \rangle|^2}{|z'|^2}. \tag{3.17}$$

Proof: Straightforward. See Appendix A.1

$\square$

So far we have a *non-invariant* metric E between curves (Equation (3.13)), which by Equation (2.1) and Equation (3.17) can be turned into an *invariant* metric by defining

$$d_{inv}(z, w) = |w'|^2 - \frac{|\langle w', z' \rangle|^2}{|z'|^2}$$

Obviously this is not symmetric, but it becomes symmetric if we normalize the vectors z and w to have norm one, in addition to zero mean.

Definition 3.2.1 *The* **Procrustes** *distance between two curves $z, w \in \mathbb{C}^n$ is*

$$d(z, w)^2 = 1 - \left|\langle \frac{z'}{|z'|}, \frac{w'}{|w'|} \rangle\right|^2 \tag{3.18}$$

By Proposition 3.2.1, the Procrustes distance between curves z and w is the squared error between the registered curves, each normalized to have zero mean and size one prior to registration:

$$d(z, w) = \min_T \left|T(\frac{z - z_0}{|z - z_0|}) - \frac{w - w_0}{|w - w_0|}\right|^2$$

Affine registration

If we try to minimize the squared error (Equation (3.14) by the more general affine transforms $T(X) = A \cdot X + \tau$, it is still possible to avoid iterative minimization methods, and find a closed form solution. However, since the affine transformations no longer have a representation in terms of complex numbers, the formulas are somewhat more complicated than in the case of scaling transforms.

Let us first introduce some notation. Suppose the curves are now $\gamma = (\gamma_1, \dots \gamma_n)$ and $\delta = (\delta_1, \dots \delta_n)$. Let γ_0 and δ_0 be their respective means, and let $\gamma' = (\gamma_1 -$

$\gamma_0 \dots \gamma_n - \gamma_0)$ and $\delta' = (\delta_1 - \delta_0 \dots \delta_n - \delta_0)$ be the zero-mean curves. Denote the coordinates of each point γ'_i by (u'_i, v'_i), and let $u' = (u'_1 \dots u'_n)$ and $v' = (v'_1 \dots v'_n)$. Let z' and w' be the analogues of u' and v', respectively, for δ'. Last, let a, b, c and d be the coefficients of the matrix A:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

Proposition 3.2.2 *With the above notations, the coefficients a, b, c, d and τ of the affine transform $T(X) = A \cdot X + \tau$ that minimizes the squared error energy* (3.11) *are given by*

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} |u'|^2 & \langle u' , v' \rangle \\ \langle u' , v' \rangle & |v'|^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \langle z' , z' \rangle & \langle z' , z' \rangle \\ \langle z' , v' \rangle & \langle w' , v' \rangle \end{pmatrix}$$

$$\tau = -(\bar{Y} - A \cdot \bar{X}).$$

Proof: See Appendix A.2.

□

Non-linear registration: Thin Plate Splines

The landmark registration under the assumption of known correspondences can be viewed as a 2D interpolation problem: we have two sets of matched landmarks $\gamma_1, \dots \gamma_n \in \mathbb{R}^2$ and $\delta_1 \dots \delta_n \in \mathbb{R}^2$, and we would like to find a transformation $T : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ which satisfies the constraints $T(\gamma_i) = \delta_i$.

Without additional constraints, there is always a trivial solution:

$$T(X) = \begin{cases} \delta_i & \text{if } X = \gamma_i \\ X & \text{otherwise.} \end{cases}$$

This solution maps each γ_i onto the corresponding δ_i, but leaves all the other points unchanged. This solution, however, is not smooth. We must find a transformation that satisfies additional smoothness constraints. A thin plate spline solution (TPS) [8] satisfies such a requirement.

The intuition behind TPS is as follows. Imagine the plane as a rubber sheet, with a square grid drawn on it. If a point z is pulled onto another location z' in the plane, an elastic deformation occurs: the lines of the grid bend in some optimal way, trying to follow the translation $z \longrightarrow z'$. From physical considerations, if $T(X) = (u(X), v(X))$ is a (non-linear) deformation of the plane, one defines the *bending energy* of T by

$$E(T) = \iint \Delta^2 T \, dX. \tag{3.19}$$

The natural mathematical formulation of the 2D interpolation problem is then to find a transformation T that minimizes the bending energy (Equation (2.12)), subject to the constraints $T(\gamma_i) = \delta_i$

To minimize the bending energy, one must compute the gradient in Equation (2.12) and solve the partial differential equation that results from setting the gradient equal to zero. The fundamental solution of this equation is the radial basis function

$$B(X) = -|X|^2 \log |X|^2 \tag{3.20}$$

which happens to satisfy $B(0) = 0$.

Clearly, the functions

$$B_i(X) = B(X - \gamma_i)$$

still minimize the bending energy, and satisfy $B_i(\gamma_i) = 0, \ i = 1, \ldots, n$.

To find the solution T of the interpolation problem, we seek T as a linear combination of the basis functions B_i plus an affine part:

$$T(X) = \sum w_i \cdot B_i(X) + A \cdot X + \tau \qquad (3.21)$$

The unknown weights w_i and the coefficients $a = (a_x, a_y)^T$, $b = (b_x, b_y)^T$ and $\tau = (\tau_x, \tau_y)^T$ of the affine component can be found by solving the *linear* system obtained by imposing the constraints $T(\gamma_i) = \delta_i$ in Equation (3.21). In matrix form, this system can be written as:

$$\begin{pmatrix} B_1(\gamma_1) & \cdots & B_n(\gamma_1) & \gamma_1^T & 1 \\ B_1(\gamma_2) & \cdots & B_n(\gamma_2) & \gamma_2^T & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ B_1(\gamma_n) & \cdots & B_n(\gamma_n) & \gamma_n^T & 1 \\ \gamma_1 & \cdots & \gamma_n & 0 & 0 \\ 1 & \cdots & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ A \\ \tau^T \end{pmatrix} = \begin{pmatrix} \delta_1^T \\ \delta_2^T \\ \vdots \\ \delta_n^T \\ 0 \\ 0 \end{pmatrix}$$

Figure 3.1 shows an example of TPS registration. In Figure 3.1 (a), the red points are the source points, and the blue points are the target points. We must deform smoothly the entire plane, so that the source points overlap exactly over the target points. The resulting deformation is shown in Figure 3.1 (b).

3.2.2 Unknown correspondences

Registration of parametric curves with *known* correspondences occurs e.g., in medical imaging, where a human expert manually annotates certain anatomical structures which should be matched across several images of the same organ (e.g., the brain). These homologous structures are called *landmarks* (hence the term "landmark registration").

However, if the contours are extracted automatically (e.g., edge detection, followed by morphological transforms such as erosion or dilation [98]), the point correspondences across images is not known. In this case, the real effort in shape registra-

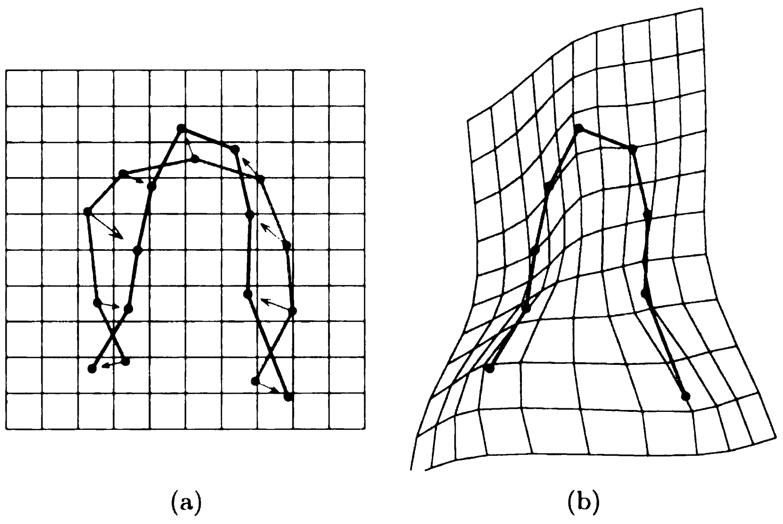


Figure 3.1: (a) Red: source points. Blue: intended target. (b) Red: the source points from (a) transformed by a TPS. They overlap exactly over the target points (blue points in (a)).

tion is in solving the point correspondence problem, either separately, or in conjunction with the registration proper. The existing proposed solutions for registration and model computation in the case of *unknown* correspondences are inherently more difficult, and more elaborated than the algorithms for landmark registration [85, 79, 69, 102, 26, 19, 113, 103, 6, 89, 34, 95, 7, 114]. It is therefore not feasible to present here all the mathematics behind these methods. Nevertheless, we shall summarize some of the main ideas.

One class of algorithms that register curves under unknown correspondences makes use of a *match matrix* $M = (M_{ij})$ [89, 34]. In the ideal case,

$$M_{ij} = \begin{cases} 1 & \gamma_i \text{ corresponds to } \delta_j \\ 0 & \text{otherwise.} \end{cases}$$

However, to establish correspondences, M can be allowed to have continuous coefficients between 0 and 1.

Rangarajan *et al.* [89] introduced the *softassign* algorithm to register two curves $\gamma_1 \ldots \gamma_{n_\gamma}$ and $\delta_1 \ldots \delta_{n_\delta}$, which have different number of points $n_\gamma \neq n_\delta$, and in which the correspondences are not known. They generalize the Procrustes metric (Section 4.1), which makes use of known correspondences, to a metric which handles the unknown correspondences by means of the match matrix M:

$$E(M, \theta, \tau, s) = \sum_{i=1}^{n_\gamma} \sum_{j=1}^{n_\delta} M_{ij} \left(\left| \frac{\sqrt{s}(\gamma_i - \mu_\gamma)}{\sigma_\gamma} - \tau - R(\theta) \frac{(\delta_j - \mu_\delta)}{\sqrt{s}\sigma_\delta} \right|^2 - \alpha \right).$$

When correspondences are known, $\mu_\gamma, \sigma_\gamma$ and $\mu_\delta, \sigma_\delta$ are the respective means and covariances of γ and δ. In the unknown correspondences case, these are computed by making use of the match matrix:

$$\mu_\gamma = \frac{\sum_{i=1} n_\gamma \sum_{j=1}^{n_\delta} M_{ij} \gamma_i}{\sum_{i=1} n_\gamma \sum_{j=1}^{n_\delta} M_{ij}}$$

$$\sigma_\gamma^2 = \sum_{i=1} n_\gamma \sum_{j=1}^{n_\delta} M_{ij} |\gamma_i - \mu_\gamma|^2$$

and similarly for δ.

The curves are then registered iteratively, in a way similar to the EM algorithm: given an estimate of the parameters, one registers γ and δ (using the Procrustes method - Section 4.1). This changes μ_γ, σ_γ, μ_δ and σ_δ. The parameters are re-estimated using the new values of the means and variances, and the procedure is repeated until convergence.

Another class of algorithms tries to establish the correspondences based on the local geometric properties of the two curves [19, 95, 26, 6]. The idea is that homologous points on the two curves must have similar *shape contexts.* To find the points with similar shape contexts, one defines an energy functional based on the local descriptors of the two curves.

For instance, when the curvature is used to describe the local geometry of two parameterized curves $\gamma(s)$ and $\delta(t)$, one can define and minimize the curvature energy

$$E(h) = \int |k_\gamma(s) - k_\delta(h(s))|^2 ds$$

where $k_\gamma(s)$ and $k_\delta(t)$ are the respective curvatures of the two curves, and h is the (unknown) diffeomorphism which provides the correspondences: $t = h(s)$. Alternative local shape descriptors are used in [6, 102].

Yet another class of techniques for establishing correspondences is the class of *iterative closest point* (ICP) algorithms [7, 114]. The idea is to match each point on the source curve γ to the closest point on the target curve δ. This establishes (non-optimal) correspondences, and a registration (with known correspondences) can be performed. Then, the closest points are computed again to re-establish correspondences, and a new registration is performed. This two-step iteration is repeated until convergence.

3.3 Distance map registration

We have seen that with parametric curves and with the squared error measure, the registration is complicated by the point correspondence problem: we must find an optimal way of linking the different parametrizations along the curves to be registered. In the discrete case, this means finding the *homologous* points on the two curves.

Using the distance map representation of a curve (Section 2.4.1, and also Equation (3.3)) and the *directed* average Hausdorff similarity measure (3.7) the point correspondence problem can be totally averted (at some cost, as we shall see). To fix the ideas, suppose γ and δ are two plane curves that we want to register. Let $T(\cdot) = T(\Theta, \cdot)$

be a 2D transformation defined by a set of parameters $\Theta = (\theta_1 \dots \theta_k)$. The set of parameters can be any one of the sets of parameters described in Section 2.1.1, or more general. We define an energy function (chamfer energy)

$$\begin{aligned} E(T) &= d_{av}(T(\gamma), \delta) \\ &= \frac{1}{|T(\gamma)|} \int_{T(\gamma)} d(T(\gamma(s)), \delta) ds \end{aligned} \tag{3.22}$$

In this equation, $d(\cdot, \delta)$ is the distance map representation of the target curve δ, and $|T(\gamma)|$ is the length of $T(\gamma)$. Division by the length $|T(\gamma)|$ penalizes the transforms that tend to map everything onto a single point. It should be noted that Borgefors [9] reports better results with the *root mean error*:

$$E(T) = \sqrt{\frac{1}{|T(\gamma)|} \int_{T(\gamma)} d(T(\gamma(s)), \delta)^2 \, ds}$$

Since we only apply T to the source curve γ, keeping δ fixed, the chamfer transform of δ can be computed once, before the minimization proper, using for instance Borgefors' algorithm [9], or the level method (Section 2.4.1). This reduces the time complexity for computing E to *linear* (in the number of points on γ). This trick only works precisely because we do not have to compute the distance transform of the moving curve $T(\gamma)$, because we are using the *directed* version of the chamfer metric (3.7). This would not be possible with any of the symmetric versions (3.5), (3.6) or (3.8) of the Hausdorff-type measures, as these entail computation of both $d(\cdot, \delta)$ *and* $d(\cdot, T(\gamma))$ (and $T(\gamma)$ varies).

Observe also, that the distance map energy (Equation (3.22)) depends on the transform T, but *does not* depend on any unknown mapping $h : \gamma \longrightarrow \delta$ that establishes point correspondences. (Compare with Equation (3.10)). The important consequence of this is that the distance map registration does not require a solution to the nontrivial problem of point correspondences. All that needs to be done now is to compute the transformation T that registers γ and δ, as

$$T_{\min} = \arg\min_T E(T).$$

Since T is uniquely identified with its parameter vector Θ, the energy (3.22) is a function of finitely many variables, and so it can be minimized using well established, off-the-shelf algorithms such as the Nelder-Mead simplex method [66, 81, 88], or - our favorite - Powell's conjugate directions method [88].

Figure 3.2 shows the results of distance map registration. In part (a), the unregistered source curve γ (blue) and the target curve δ (red) are shown. The initial value of the energy (Equation (3.22)) is $E = 2584$. Minimization using *scaling transforms* yields an energy of $E = 1563$. The results of the registration are shown to the right, in Figure 3.2 (b).

It is interesting to see what happens if we use e.g., *affine transforms* instead of scaling transforms. Since the affine transforms allow for more variation in shape (e.g., shear), the two curves can come closer to each other, so it is expected that the minimum in Equation (3.22) be smaller in the case of affine transforms. This is indeed the case in practice. If we register the same two curves from Figure 3.2 using affine transforms, we get a value $E = 989$, significantly lower than $E = 1563$ in the case of scaling transforms. The affine counterpart of Figures 3.2 (a) and (b) is shown in Figures 3.2 (c) and (d). The two curves come much closer to each other (Figure 3.2 (d)) than in the case of scaling transforms (Figure 3.2 (b)), but the results are hardly what was intended. This shows the importance of two factors:

- choice of the appropriate class of transformations.
- importance of the non-symmetry of the chamfer metric (3.7) and (3.22). We have to use the non-symmetric version of the distance between $T(\gamma)$ and δ to

reduce the time complexity, but the price we pay for this is the possibility of artificially good, but disproportionate alignments of the source onto the target, like the on in Figure 3.2 (b).

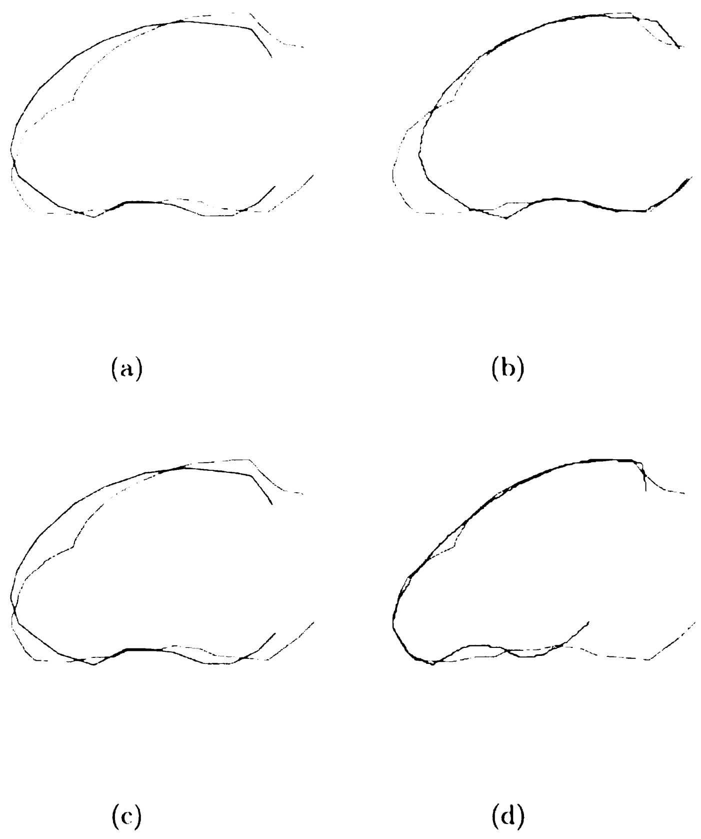


Figure 3.2: (a),(b): Distance map registration with *scaling* transforms. The measure of fit is $E = 1563$. (c), (d): Distance map registration with *affine* transforms. Measure of fit: $E = 989$. Left column: unregistered. Right: registered. Blue: source. Red: source.

3.4 Semiaxes registration

We describe in this section a novel registration algorithm with unknown correspondences, most suitable for objects with one axis longer than the other. We call this *semiaxes registration.*

Let $\gamma = ((\gamma_1, \dots, \gamma_n)$ and $\delta = ((\delta_1, \dots, \delta_m)$ be two sets of points (possibly, but not necessarily, discretized continuous curves). If each of γ and δ have roughly the shape of an ellipse, then it would make sense to try to align the two sets of points by aligning the underlying ellipses. Here is the algorithm:

Normalization. First we normalize the two sets to have zero mean and size one. The normalized sets of points are:

$$\gamma \leftarrow \frac{\gamma - \overline{\gamma}}{|\gamma - \overline{\gamma}|} \qquad \delta \leftarrow \frac{\delta - \overline{\delta}}{|\delta - \overline{\delta}|}$$

Rotation. Compute the unit eigenvectors (e_1, e_2) of the 2×2 covariance matrix of the set γ, with corresponding eigenvalues $\lambda_1 > \lambda_2$. Similarly for δ, let (f_1, f_2) be the unit eigenvectors, with corresponding eigenvalues $\mu_1 > \mu_2$. We choose the signs of the eigenvectors so that both the (e_1, e_2) and the (f_1, f_2) frames have positive orientation. In general, computation of eigenvalues and eigenvectors is non-trivial, but powerful iterative algorithms exist [88]. However, in dimension 2 it is trivial to derive closed form formulas for both the eigenvalues and the eigenvectors, due to the fact that the characteristic polynomial of the 2×2 covariance matrix is quadratic. Thus, computation of the eigenvectors and the eigenvalues is done in constant time, if the covariance matrix is given. Computation of the covariance matrix is obviously linear in the number of points in the set.

If the eigenvectors play the role of semiaxes, then we must align them, so we must compute the rotation (around the origin, as γ and δ have zero mean now), which brings e_1 over f_1 and e_2 over f_2. This rotation exists, because the two frames are orthonormal and have the same (positive) orientation.

Of course, when computing the eigenvectors of γ, say, there are two choices in choosing the positive-orientation eigenvector frame: $\pm(e_1, e_2)$. Each of these frames must be rotated over each of $\pm(f_1, f_2)$, resulting in four rotations

$R_1, \ldots R_4$. We have to choose the best one. To define "best", we can use any metric between sets of points in $\mathbb{R}^2$ that does not assume point correspondences (e.g., the Hausdorff distance (3.6))

The transform T that aligns the source set γ onto the target set δ is a composition of translation, scale and rotation, possibly also composed with a symmetry. It is, therefore, a *scaling* transform (possibly with negative scale).

1. **Advantages:**

   - does not assume correspondences, nor does it try to find them.
   - speed and simplicity: no minimization is required.
   - can be combined with distance map registration (Section 3.3), by providing a very good initialization to the distance map minimization procedure.

2. **Limitations:**

   - ambiguity (ill-posedness) increases when registering *circular* sets (i.e., when the eigenvalues are approximately equal). This would be the case, for instance, when trying to align two circular saw blades: even if they were identical, the teeth might be offset by a random amount.
   - there is no *obvious* metric with respect to which the aligning transform T is optimal.

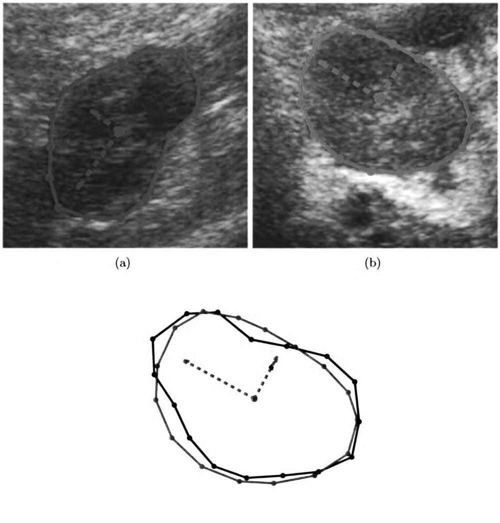


Figure 3.3: Semiaxes alignment of two contours. (a) and (b): Two contours and their corresponding eigenvectors drawn at their respective centers. Each eigenvector is proportional to its own eigenvalue. (c) Alignment. The blue contour (source) was brought over the red contour (target), by aligning the eigenvectors.

3.5 Summary

Here are the main points of this chapter:

- Given two curves γ and δ we would like to measure their *shape* similarity. To this end, we need a metric $d_{inv} = d_{inv}(\gamma, \delta)$ *invariant* under e.g., scaling transforms. It is often easier to define a non-invariant metric first $d = d(\gamma, \delta)$, and turn it into an invariant metric by minimizing the energy functional defined by

$$E_{\gamma,\delta}(T) = d(T(\gamma), \delta)$$

  where T ranges over a set of parametric transformations (Section 2.1.1), and E is an optimality criterion (such as the squared error energy (Equation (3.10)), or the distance map energy (Equation (3.22))). Minimizing the energy functional $E(T)$ is called curve registration. The invariant shape metric can then be defined by

$$d_{inv}(\gamma, \delta) = \min_T E_{\gamma,\delta}(T)$$

- Curves extracted from digital images can be represented
  - explicitly, as parametric curves: $\gamma = \gamma(t)$ (continuous case), or as n-dimensional complex vectors $z = (z_1, \ldots, z_n) \in \mathbb{C}^n$. Registration of parametric curves depends on the point correspondences between the curves. These can be
    * known (Section 3.2.1). Closed form formulas exist in the case of scaling and affine registration (Propositions 3.2.1 and 3.2.2) as well as in the case of TPS registration (Equation (3.21)).

    * unknown (Section 3.2.2). Registration is much more difficult than in the case of known point correspondences.

- implicitly, as the zero set of their distance map.

    * distance map registration avoids establishing point correspondences, but the lack of symmetry may have undesirable consequences (Section 3.3). The minimization of the optimality criterion (Equation (3.22)) is done iteratively, each iteration having complexity linear in the number of points in the source set.

    * semiaxes registration is an efficient, albeit suboptimal, registration method that also does not assume correspondences. The transform that registers the two sets of points can be computed in closed form. This is one contribution of our thesis and will be our method of choice in our experiments.

Chapter 4

Model Learning

Our first step in building a segmentation system with shape memory is to construct a *model* of the targeted object. To this end we assume the boundaries of the object of interest were manually outlined by a human expert in several images containing various instances of the object. These curves could be, for instance, contours of different kidneys (or any other organ, for that matter) in ultrasound images from different patients. Throughout this chapter, the training set will be denoted by $\{\gamma_1, \ldots, \gamma_n\}$.

Learning a model from a set of training samples amounts, in general, to computing statistics of the training set relevant for the problem at hand. For instance, one of the simplest cases is to compute some sort of mean of the training samples, which could then serve as a template that could be fit to an image in search for a certain object. However, computing the mean is not always straightforward, for the very same reasons outlined in the beginning of Section 3.1. In this chapter we show how to compute a *mean shape* when the notion of mean makes sense at least mathematically, in the case of known point correspondences (Section 4.1), then we show how we can extend the notion of shape model to the case of unknown point correspondences.

Before we proceed, we must note that the mean is in some sense a minimalistic shape model, can only be used when it describes sufficiently well the variation of the training samples. For instance, one may be interested, besides the mean, in the *modes of variation* of the shape distribution. In such cases more sophisticated models are required, such as a good approximation of the very distribution of the training samples.

4.1 The Procrustes Model

Let us assume known point correspondences, in the most favorable situation, and see how we can compute a mean shape, and eventually a probability distribution of the training shapes. The assumption of known point correspondences entails, of course, that all γ_i have the same number of points, so at least mathematically, computation of the mean $(\sum \gamma_i)/n$ makes perfect sense.

Recall, however, from Section 2.1 that when comparing shapes we must factor out the extrinsic differences, which means mathematically that we must register the curves by finding a transform T optimal in some well defined sense. In the case of known correspondences, it is natural to use the squared error energy (L^2 distance, Equation (3.14)) as the optimality criterion for T. By contrast with the problem of two-curve registration (Section 3.2.1), we are now faced with the more difficult problem of *simultaneous* registration of all the training samples $(\gamma_1, \dots, \gamma_n)$, termed *generalized Procrustes analysis* by Dryden and Mardia [30]. We shall assume the registration is done using scaling transforms:

$$T(\omega) = \lambda\omega + \tau \qquad \forall\omega \in \mathbb{C}$$

Following [30], the problem of simultaneous registration can be formulated by seeking scaling transforms $(T_1, \dots, T_n)$ (one for each γ_i) which minimize

$$E(T_1, \ldots, T_n) = \sum_{i<j} |T_i(\gamma_i) - T_j(\gamma_j)|^2 \tag{4.1}$$

subject to the constraint

$$|\overline{\gamma}| = 1$$

where $\overline{\gamma} = \sum T_i(\gamma_i)/n$ is the mean of the *transformed* training samples. By minimizing the energy (4.1), we effectively bring all training samples together in a global way. The constraint $|\gamma| = 1$ is necessary to avoid the trivial solution $T_i = 0\ \forall i$.

Once the optimal transforms $(T_1, \ldots, T_n)$ have been found, then one can compute the mean shape as

$$\overline{\gamma} = \frac{1}{n} \sum T_i(\gamma_i) \tag{4.2}$$

Remark 4.1.1 *It turns out (Result 5.2 in [30]) that the arithmetic mean $\overline{\gamma}$ of the registered samples is the shape closest to all the training shapes in the sense of the invariant Procrustes metric* (3.18). *That is, if $d(\cdot, \cdot)$ is the Procrustes metric, then*

$$\mu = \arg \min_{|\mu|=1} \sum d^2(\gamma_i, \mu) \tag{4.3}$$

In fact, Dryden and Mardia take Equation (4.3) *as the definition of the* full Procrustes mean. *Equation* (4.3) *does not involve component-wise operations on the training samples, so it allows generalization to the case of unknown point correspondences.*

Unlike the case of two-curve scaling registration (*ordinary Procrustes analysis*), there is no closed form expression for the optimal set of transforms T_i. An iterative algorithm is presented in [30]. Cootes and Taylor [22] propose a more practical one: Select one training sample as reference curve, and register by similarity transforms all remaining training samples to the reference curve (Proposition 3.2.1). Compute the mean of the aligned curves and repeat the procedure with the newly computed mean as reference curve, until convergence.

Yet another popular (but sub-optimal) Procrustes algorithm is as follows:

- Normalize each training sample γ_i to have zero mean and unit size.
- Pick one of them, say γ_1 as reference curve and rotate each of the remaining γ_i so as to minimize the squared error $|\gamma_i - \gamma_1|$.

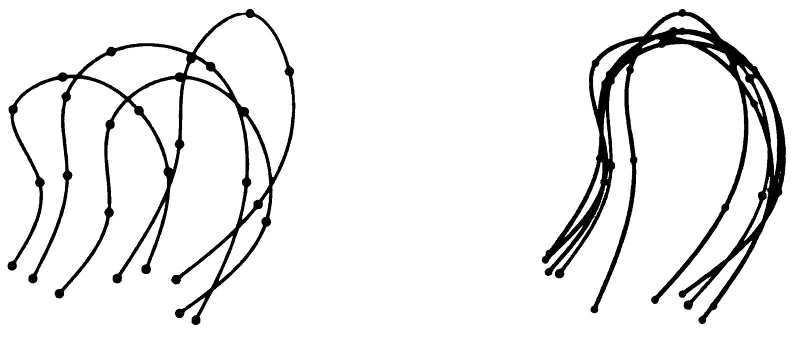


Figure 4.1: Procrustes model. (a) Training samples (blue) with landmarks (red). (b) Registered training samples (blue) and their mean (purple).

Having registered the training samples in some globally optimal way, we can compute not only the mean, but we can actually estimate the distribution of the given shapes by using any of the standard density estimation methods such as PCA (termed *point distribution model* by Cootes and Taylor in [22]), or kernel-based methods such as Parzen windows [33].

4.2 Average Chamfer Transform Model

Both the Procrustes and the Point Distribution models require known point correspondences. Our main application is directed at border detection in kidney ultrasound images. Empirically, our experience is that there are *not enough anatomical kidney landmarks identifiable in ultrasound images, that are consistent across a significant number of training samples.* Always, when building a shape model there has to be some statistical regularity of the training shapes, or else, the model would be too coarse.

We found that it is more useful to build a model that only registers the kidney training samples based on their overall shape, without using any point correspondences, rather than try to look for point correspondences when there are none.

Specifically, let $(\gamma_1, \ldots, \gamma_n)$ be a set of training samples [1], e.g., contours manually drawn by a human expert. Our model will provide

- *a base curve* - some sort of average shape,
- a way to measure how similar a test contour δ is to *all* of the training samples, simultaneously.

Let $d(\cdot, \cdot)$ be a similarity measure (distance) between shapes. We have seen (Section 2.1) that either d must be invariant to scaling transformations to begin with (e.g., the curvature distance in Section 2.5), or, if d is not invariant, then the shapes must be *registered* before measuring the distance between them (e.g., chamfer transform, Hausdorff distance, etc.).

In the case of an *invariant* distance d, we can build a model as follows. If δ is a test shape, and γ_i is any of the training samples, then $d(\delta, \gamma_i)$ measures the similarity between δ and that particular γ_i. Naturally, then, a measure of similarity of δ against not only a single training shape, but against *all* training shapes, is the average of these distances after all training shapes:

$$E(\delta) = \frac{1}{n} \cdot \sum_i d(\delta, \gamma_i)$$

[1] For reasons outlined in the beginning of Section 2.3, we pre-process these contours by computing a cubic interpolation spline on the control points marked by the human expert.

As observed in Remark 4.1.1, we can take as *base curve* the shape closest (in the sense of E) to all the training samples:

$$\mu = \arg\min_{\delta} E(\delta)$$

If the metric d is not invariant, we proceed in the same manner, except that we register the shapes that must be compared.

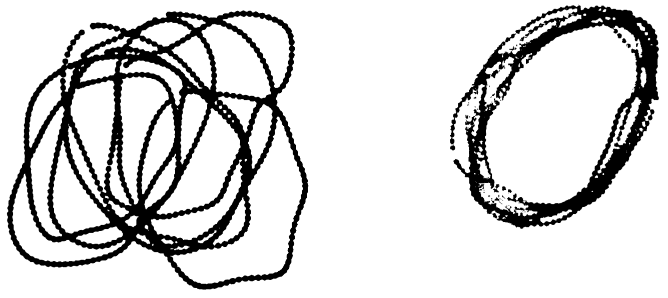

(a) (b)

Figure 4.2: Semiaxes registration of training samples. It does not assume or require known point correspondences. (a) Unregistered training shapes (green), and randomly chosen target (blue). (b) Registered training shapes (red).

To fix the ideas, let

$$C\gamma_i(p) = \min_t |p - \gamma_i(t)|$$

be the chamfer transform of γ_i, and let d be the induced distance:

$$d(\delta, \gamma_i) = \int_\delta C\gamma_i(\delta(s))ds$$

We randomly designate one training sample as target, and perform a *semiaxes registration* (Section 3.4) of any of the remaining training samples onto the target. For simplicity, we shall denote the *registered* training samples also by γ_i, as the originals. We build the chamfer transform C_{γ_i} on each of the *registered* training shapes, and compute the average:

$$\overline{C} = \frac{1}{n} \sum_i C_{\gamma_i}$$

For each *point* (x, y), $\overline{C}(x, y)$ represents the *average* distance from (x, y) to the training shapes. If δ is any curve (spline or otherwise), define the distance from δ *to the training set* by

$$\overline{C}(\delta) = \int_\delta \overline{C}(\delta(t))dt \tag{4.4}$$

The quantity $\overline{C}(\delta)$ measures how close a test shape δ is to all the training samples. We seek a curve μ with the same topology as the training samples, which among all such curves, is closest on average to all the training samples. That is, we seek a curve

$$\mu = \arg\min_\delta \overline{C}(\delta) \tag{4.5}$$

Now, if $\overline{C}$ were the chamfer transform of some curve[2] δ_0, then $\min_\delta \overline{C}(\delta)$ would be zero, and would be achieved for $\mu = \delta_0$. In general, however, the average distance map $\overline{C}$ is not the chamfer transform of any curve, but it still has the shape of a chamfer. See Figure 4.3. We can still compute a curve μ as in Equation (4.5) by tracking the bottom of $\overline{C}$.

So far we have (a) a functional $\overline{C}$ - the *average chamfer transform* - which measures for each curve δ how close that curve is to *all* the training samples (Equation (4.4), and (b) curve μ closest to all the training samples, in the sense of (4.5).

[2]Non-degenerate, simple and closed.

Note, however, that $\overline{C}$ measures *spatial proximity*, rather than shape similarity, because $\overline{C}$ is not invariant under similarity transforms. Loosely speaking, this requires that we must first register a curve δ to the training samples, before computing $\overline{C}(\delta)$. The most natural way to make sense of this non-rigorous statement is to register δ to the base curve μ. Thus, if we denote by $T_{\delta,\mu}$ the transform that aligns δ onto the base curve μ then we define the global similarity between δ and the training samples as

$$E(\delta) = \overline{C}(T_{\delta,\mu}(\delta)) \tag{4.6}$$

In summary, after computing the average chamfer transform $\overline{C}$ of all the *registered* training samples, we compute the base curve μ by tracing the bottom of $\overline{C}$. Then, given a test curve δ, we register δ to μ (by a transform $T_{\delta,\mu}$), and only then compute the similarity $\overline{C}(T_{\delta,\mu}(\delta))$.

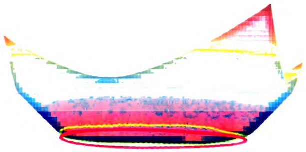
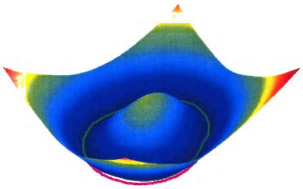



Figure 4.3: Two views of the average chamfer transform $\overline{C}$, and the base curve μ (purple). The green curve is the trace of the bottom of $\overline{C}$, and μ is its projection onto the xy plane.

Part II

Shape Detection

Chapter 5

Active Contours

This thesis is, in essence, about border detection using active contours, so we review the literature in some detail in this chapter. Active contours have been used extensively in many computer vision problems, particularly in boundary detection and motion tracking, with relative success, especially in medical imaging. Almost two decades after the introduction of the concept by Kass, Witkin and Terzopoulos [63], a great variety of different "species" of snakes have been developed. One can distinguish two major categories: *variational models*, and *geometric models.*

5.1 Variational Models

Chronologically, the variational models occurred first [63], and are based on the observation that if we want a curve[1] γ to be aligned along the edges in an image, then γ must accumulate the largest amount of gradient. In other words, γ must minimize an *image energy* defined as:

$$E_{img}(\gamma) = -\int_0^1 |\nabla I| ds \qquad (5.1)$$

[1]Throughout this chapter, γ is a closed curve.

or, more generally, for any decreasing function $g = g(\xi)$,

$$E_{img}(\gamma) = \int_0^1 g(|\nabla I|)ds \tag{5.2}$$

If smooth minima are sought, then an extra energy term must be added into the equation[2], to penalize for the formation of discontinuities:

$$E_{int}(\gamma) = \int_0^1 |\gamma'|^2 + |\gamma''|^2 ds \tag{5.3}$$

The original Kass, Witkin and Terzopoulos active contour model includes a third energy term - the external energy E_{ext}, which can incorporate e.g., prior knowledge about the location or the shape of the boundaries of interest. One then seeks minima of an energy functional of the form

$$\begin{aligned} E(\gamma) = \alpha \cdot & \underbrace{\int_0^1 g(|\nabla I|)ds}_{E_{img}} \\ + \beta \cdot & \underbrace{\int_0^1 |\gamma'|^2 + |\gamma''|^2 ds}_{E_{int}} \\ + E_{ext} & \end{aligned} \tag{5.4}$$

Minimization of the energy (5.4) is done through variational principles which entails that local minima are found. Consequently, initialization near the boundaries of interest is required. The minima (and maxima, and the saddle points) of the energy (5.4) are critical points of E, so they are curves which satisfy the equation $dE/dt = 0$, where the derivative of E is computed as follows. Let $\gamma : [0, 1] \longrightarrow \mathbb{R}^2$ be an arbitrary snake, and let $\tilde{\gamma}_t$ be a 1 parameter deformation of γ. That is,

[2]This technique of adding an extra term, for constraining the solutions is called Tykhonov regularization. See for instance http://en.wikipedia.org/wiki/Ridge_regression and references therein.

$$\tilde{\gamma} : [0, T] \times [0, 1] \longrightarrow \mathbb{R}^2, \qquad (t, s) \longrightarrow \tilde{\gamma}(t, s) := \tilde{\gamma}_t(s)$$

$$\tilde{\gamma}_0 = \gamma$$

The derivative of E at γ is then *defined* as

$$\begin{aligned} \frac{dE}{dt}(\gamma) &= \left.\frac{d(E(\tilde{\gamma}_t))}{dt}\right|_{t=0} \\ &= \left.\frac{\partial E(\tilde{\gamma}(t, s))}{\partial t}\right|_{t=0} \end{aligned} \tag{5.5}$$

In general, computation of the derivative (5.5) is not always straightforward, depending on the definition of E. The resulting system of equations $dE/dt = 0$ is called the *Euler-Lagrange* equations of E, and, in our case, it is a 2-equation system of ordinary differential equations (ODE), because the variable $\gamma = (x, y)$ is a function of only one real variable [3]. For the explicit derivation of the Euler-Lagrange equations in the case of snakes and a numerical implementation of a solution, see the original paper [63].

This model is known to have a number of shortcomings, some of which are:

- The snake energy (5.4) depends on the parametrization of γ. One could change the energy without changing the shape of the snake, simply by changing the parametrization.

- Numerical stability: as the snake evolves, the points on it can slide along the snake coming too close together in some regions (e.g. being "attracted" by an energy pit), while leaving the rest of the snake sparse. For numerical stability, the points on the snake must be maintained relatively equidistant, which requires meticulous point insertions and deletions at each snake iteration.

[3]In general, the Euler-Lagrange equations is are partial differential equations (PDE).

- In homogeneous regions (i.e. where $|\nabla I| \approx 0$), the snake does not move, as the forces acting upon it are (close to) zero.

- The snake does not change topology. Although in general this is viewed as a negative feature, it can become useful when detecting objects whose topology is known a priori, such as in medical imaging. See Section 5.3 for an example.

The active contour model underwent major improvements (themselves variational models) e.g., in [20, 2, 110, 112, 73], trying to address the above, and other issues.

5.2 Geometric Models

The geometric models were introduced independently in [14] and [71] based on front propagation theory [82, 83] and were a revolutionary way of viewing active contours. These authors proposed a non-variational approach, namely mean curvature motion along the normal direction. More precisely, let $\gamma_t(s) = \gamma(t, s)$ be the evolving curve, at moment t. Here s is the parameter along the curve and t shows the dependence of the curve on time. Let ν be the *inward* unit normal along the curve. See Remark 2.4.1 for this choice of ν. The requirement that any point on γ_t move along the normal is described mathematically by the following differential equation:

$$\partial_t \gamma = F \cdot \nu \tag{5.6}$$

where $F : \mathbb{R}^2 \longrightarrow \mathbb{R}$ is a real-valued function (speed). Note that Equation (5.6) does not necessarily arise as an Euler-Lagrange equation of some energy functional, but it is rather simply a geometric requirement. The evolution of γ is controlled by a judicious choice of the speed function F, based on the following observation: when F is small, γ changes very slowly, and when F is large, γ changes rapidly. Though not mandatory, a good (and well studied) choice for the speed is to take F a function on the curvature k of γ, i.e., $F = F(k)$. The first and simplest example turns out to be surprisingly important and well understood:

Example 5.2.1 *$F = c + k$ where $c > 0$ is a real constant. The corresponding flow is*

$$\partial_t \gamma = (c + k) \cdot \nu$$

The $c \cdot \nu$ component moves the curve inward ($c > 0$) or outward ($c < 0$), whereas the $k \cdot \nu$ component has been shown to shorten as well as smooth out γ, and to decrease the total curvature of γ [97].

Let us now discuss the implementation of geodesic models. As explained in Section 2.2, there are two ways to represent the curve γ_t: explicitly, as $\gamma_t = \gamma_t(s) = (x_t(s), y_t(s))$, or implicitly, as the level set of some surface. In the first case, the implementation is straightforward: discretize γ, compute the (unit) normal ν at each point and move the point in the direction of ν. However, this naive implementation is known to be very unstable numerically. Fortunately, a far more elegant, powerful and stable method arises if, at any time t during the evolution, we view the active contour as the zero level-set of an (also evolving) surface $\phi_t(x, y) = \phi(x, y, t)$. That is, $\gamma(s, t) = (x(s, t), y(s, t))$ satisfies the equation

$$\phi(x, y, t) = 0 \quad \forall t \geq 0 \tag{5.7}$$

We now have a surface $\phi_t(x, y)$ whose zero-level set γ_t must evolve according to (5.6). Naturally, the question is how must ϕ_t evolve in order that γ_t satisfy (5.6)? By differentiating (5.7) with respect to t, we get

$$\begin{aligned}
0 &= \partial_t \phi(x, y, t) \\
&= \partial_x \phi \cdot \partial_t x + \partial_y \phi \cdot \partial_t y + \partial_t \phi \\
&= \langle \nabla \phi, \partial_t \gamma \rangle + \partial_t \phi \\
&= F \cdot \langle \nabla \phi, \nu \rangle + \partial_t \phi \qquad (by\ (5.6)) \\
&= F \cdot \langle \nabla \phi, \frac{\nabla \phi}{|\nabla \phi|} \rangle + \partial_t \phi \\
&= F \cdot |\nabla \phi| + \partial_t \phi \qquad (5.8)
\end{aligned}$$

In this last equality we have used the fact that, since γ_t is a level curve of ϕ_t, its unit normal ν is given by $\nu = \nabla\phi/|\nabla\phi|$ (as the gradient of a surface is orthogonal to the level curves of that surface). We have proven then that if we want the zero-level curve γ_t of a time-dependent surface ϕ_t to move along its own normal with speed F (eq. 5.6), then the surface itself should evolve according to the following equation:

$$\partial_t \phi = -F \cdot |\nabla \phi| \qquad (5.9)$$

$$\phi(\cdot, 0) = \phi_0 \qquad (5.10)$$

where the initial condition ϕ_0 is some surface whose zero-level set is the initial snake γ_0. A good candidate for this is the *signed* distance map of γ_0 (see Section 2.2, particularly Remark 2.4.1).

Equation (5.9) falls in the category of *Hamilton-Jacobi* equations (H-J), i.e., it is an equation of the form

$$\partial_t \phi = H(\partial_x \phi, \partial_y \phi)$$

for some function H (called *Hamiltonian*). Specifically, in the case of (5.9)

$$H(u, v) = -F \cdot \sqrt{u^2 + v^2}$$

The existence of solutions of H-J equations, their properties, as well as stable numerical schemes for solving them, are highly non-trivial mathematical subjects. Thanks to the deep results of Lions and Crandall [23], Osher and Shu [84], Osher and Sethian [83], Sethian [97] (and references therein), there are known numerically stable schemes that can be used to solve H-J equations. Efficient algorithms such as the *fast marching* method [96] and the *narrow band* method [1] have also been found, and make the level-set snakes suitable for real-time applications.

5.2.1 Geodesic Active Contours

A milestone in the theory of active contours was the introduction of *geodesic active contours* by Caselles, Kimmel and Sapiro [15], which provide a unifying view of the variational and non-variational models, and are arguably the state-of-the-art in active contours. Specifically, these authors set out to minimize the energy (5.4), less the $|\gamma''|^2$ term and the E_{ext} term:

$$E(\gamma) = \alpha \cdot \int_0^1 g(|\nabla I|)ds + \beta \cdot \int_0^1 |\gamma'|^2 ds \tag{5.11}$$

Next, it is shown via a clever application of the Maupertuis' principle [31], that minimization of the simplified energy ((5.11)) is equivalent to the minimization of the following energy[4]:

$$E_{gac}(\gamma) = \int_0^1 g(|\nabla I|) \cdot |\gamma'| ds \tag{5.12}$$

The gain is that the Euler-Lagrange equations of E_{gac} describe a motion in the

[4]This is not to say that the two energies are equal! They just have the same minima

normal direction of γ, as in the case of geometric models (Section 5.2):

$$\partial_t \gamma = g(I) \cdot k \cdot \nu - \langle \nabla g, \nu \rangle \cdot \nu \tag{5.13}$$

where k is the curvature along γ, and as before, ν is the unit inward normal to γ. In order to make the snake evolve through regions of zero gradient, an extra constant is introduced in (5.13), inspired by the flow in Example 5.2.1:

$$\partial_t \gamma = g(I) \cdot (c + k) \cdot \nu - \langle \nabla g, \nu \rangle \cdot \nu \tag{5.14}$$

Clearly, this is, a variational model (at least up to Equation (5.13)), but also a particular case of geodesic model with speed $F = g(I) \cdot (c + k) + < \nabla g \,, \nu >$. Consequently, as with any geodesic model, the evolution of γ can be carried out numerically using a level-set implementation. The surface evolution equation corresponding to (5.14) is:

$$\partial_t \phi = g \cdot (c + k) \cdot |\nabla \phi| + \langle \nabla g, \nabla \phi \rangle \tag{5.15}$$

$$\phi(0, \gamma) = \phi_0(\gamma) \tag{5.16}$$

where ϕ_0 is the signed distance map[5] of the initial snake. As for the function g, a very popular choice is

$$g(|\nabla I|) = \frac{1}{1 + |\nabla I|^2} \tag{5.17}$$

Geodesic active contours are known to work better in many cases than both their ancestors, and are arguably the state of the art in active contours. Being implemented using the level sets approach, they can seamlessly change topology. These snakes

[5]Following the convention in Remark 2.4.1: negative outside the snake, positive inside.

were developed under the assumption of uniform boundaries, and if this condition is violated, the snake will leak across the boundary, through the regions of low gradient. As with any geometric model, one major difficulty is the stopping criterion, because Equation (5.9) can be evolved for arbitrarily large t, and as long as the speed F is not absoluteely 0, the snake will continue to move (albeit at a low speed) right through the boundaries it is supposed to detect. A small, but non-zero speed will slow down the snake evolution, but will not stop it completely, so if given a chance, a geometric model could theoretically evolve forever.

5.2.2 The Chan-Vese Model and the Mumford-Shah Model

No discussion on active contours and segmentation would be complete without the Mumford-Shah model and one of its particular instances - the Chan-Vese model.

In a milestone paper [80] David Mumford and Jayant Shah introduce a very powerful mathematical framework for image segmentation. To fix the ideas, let I be the image, defined on a rectangle R. Informally, a segmentation of I is a piecewise constant approximation of I. More formally, we seek a decomposition of R into a disjoint union

$$R = R_1 \sqcup R_2 \cdots \sqcup R_n \sqcup \Gamma$$

and a piecewise smooth function f defined also on R which approximates I in some optimal sense, and which is nearly constant on each component R_i. Γ is simply the union of the boundaries of the R_is. Knowing Γ is equivalent to knowing the partition. As optimality criterion they propose the following energy functional (subsequently termed the *Mumford-Shah functional*):

$$E(f,\Gamma) = \mu \iint_R (f - I)^2 + \iint_{R\backslash\Gamma} |\nabla f|^2 + \nu|\Gamma| \qquad (5.18)$$

Note that both the approximation f *and* the partition Γ are unknown. The first term requires that f approximate the image I as much as possible in L^2 norm. The second term forces the function f to be as flat as possible on each of the R_is. The last term (the length of Γ) forces Γ to be as short as possible, in order to rule out too fine a partition. After all, in an extreme (but useless) case, we could define a one-pixel R_i at each pixel.

The mathematics of this functional far exceeds the scope of this thesis. It is highly non-trivial, using powerful techniques from the theory of partial differential equations, and involves computation of the first and second variation, and proving results on the existence of the solutions.

There is one notable particular case of the general Mumford-Shah theory introduced by Tony Chan and Luminita Vese in [16]. In the simplest case, the Chan-Vese active contour model tries to segment a bi-modal image u_0 by trying to find the two gray levels c_1 and c_2 as well as the boundary Γ between regions of different intensities, by minimizing the energy functional

$$E_{CV}(c_1, c_2, \Gamma | u_0) = \int_{int(\Gamma)} |u_0(x) - c_1|^2 dx + \int_{ext(\Gamma)} |u_0(x) - c_2|^2 dx + \nu \cdot |\Gamma| \quad (5.19)$$

where ν is a constant. Embedding Γ as the zero level curve of a surface ϕ, such that

$$int(\Gamma) = \{x | \phi(x) < 0\}$$
$$ext(\Gamma) = \{x | \phi(x) > 0\}$$

then the Chan-Vese functional can be re-written in terms of c_1, c_2 and ϕ as

$$E_{CV}(c_1, c_2, \phi | u_0) = \int_\Omega |u_0 - c_1|^2 H(\phi) + \int_\Omega |u_0 - c_2|^2 (1 - H(\phi)) + \nu \int_\Omega |\nabla H(\phi)| \tag{5.20}$$

where H is the Heaviside function:

$$H(x) = \begin{cases} 1 & if x > 0 \\ 0 & if x < 0 \end{cases}$$

Taking partial derivatives with respect to c_1, c_2 and ϕ, we get the Euler-Lagrange equations:

$$c_1 = \frac{\int_\Omega u_0 H(\phi)}{\int_\Omega H(\phi)} \tag{5.21}$$

$$c_2 = \frac{\int_\Omega u_0 (1 - H(\phi))}{\int_\Omega (1 - H(\phi)} \tag{5.22}$$

$$\partial_t \phi = \delta(\phi)\Big(\nu \cdot div\Big(\frac{\nabla \phi}{|\nabla \phi|}\Big) - |u_0 - c_1|^2 + |u_0 - c_2|^2\Big) \tag{5.23}$$

where δ is the Dirac delta function.

Note that the Chan-Vese model is formulated entirely without any reference to the gradient ∇u_0 of the image. It has also been observed that it is a very particular case of the Mumford-Shah model, in which the image is assumed piecewise constant, with only two gray levels.

Segmentation of the image u_0 amounts to finding the gray levels c_1, c_2, and the surface ϕ (whose zero level set is ultimately sought), solutions of the system of Equations (5.21), (5.22) and (5.23). When the unknowns c_1, c_2 and ϕ have been found, the segmented image is

$$u = c_1 \cdot H(\phi) + c_2 \cdot (1 - H(\phi))$$

i.e., the segmented image u consists of pixels of intensity c_1 inside the detected boundary $\Gamma = \{\phi(x) = 0\}$ and pixels of intensity c_2 outside Γ.

From an implementation standpoint, observe that while it is trivial to compute c_1 and c_2 (given ϕ) as the average intensities over the interior and the exterior of Γ, respectively, the evolution Equation (5.23) poses major numerical problems, requiring a discretization that produces a *stable* numerical scheme. We use the scheme proposed in the original paper [16] [6]. To our knowledge[7], the convergence of the numerical scheme to the actual solution ϕ has not been shown.

Examples of Chan-Vese segmentation are shown in Section 8.2.1 (ultrasound) and in Section 8.4 (synthetic noisy images).

5.3 Parametric Snakes

We discuss in this section a sub-category of variational active models, whose shape is controlled by a relatively small number of parameters. Perhaps the most popular snakes in this sub-category are the B-spline snakes (or B-snakes) [27, 29, 5, 39, 11], but there are other types as well. We term them all as *parametric snakes.*

There are important domains, such as ultrasound, which provide very irregular visual support for the boundaries of the object of interest, while the true boundaries are known a priori to be smooth. See Figure 5.1. Ultrasound is one of the toughest domains in computer vision, and still presents great challenges to just about any segmentation or border detection algorithm, due to the noisy, specular nature of the ultrasound images and incomplete data, with misleading visual support. The edges

[6] We are deeply grateful to Luminita Vese for the help with the implementation of the Chan-Vese algorithm.

[7] And to Luminita Vese's knowledge.

in such images are usually weak, spurious, and have non-uniform magnitude.

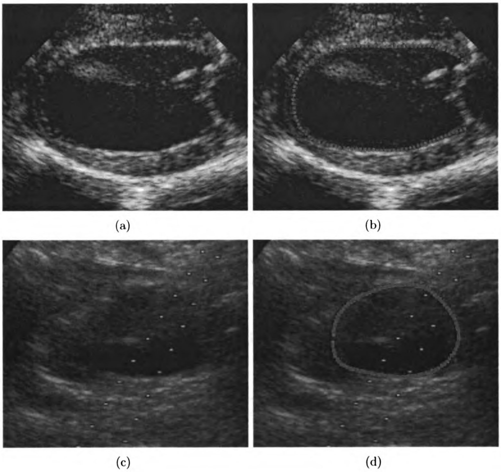


Figure 5.1: Sample ultrasound images. (a) and (b) heart left ventricle (dog). (c) and (d) kidney (human). The green curve in (b) and (d) is the true boundary (*ground truth*) and represents the goal of any border detection algorithm. Observe in the clear images (a) and (c), that the edges have non-uniform intensity, and are discontinuous. In particular, in (b), the apex of the ventricle (left side of the green curve) is some $10-15$ pixels off the visible boundary (misleading edges), passing through a region of incomplete data. Similar considerations are valid for the kidney images (c) and (d).

Under these extreme conditions, classical variational models usually follow the data too closely, and the detected boundaries do not meet the smoothness requirement. Geometric models pose additional problems. Their signature feature - the ability to seamlessly change topology - is detrimental in ultrasound, because the snake can and will detect artifacts, muscles or other visually salient structures which are not in fact part of the object of interest. Another problem with geometric snakes - perhaps the most stringent under the conditions of non-uniform boundaries - is the stopping criterion: the snake may stop prematurely in some regions, while leaking through the boundaries in other places. These problems are illustrated in Figure 5.3 (b) [8]. The snake stops around the inner structure (papillary muscle) while it leaks through the valve, and near the apex. More often than not, a single set of parameters, regardless of their values, simply cannot accommodate both strong and very weak boundaries along the same contour.

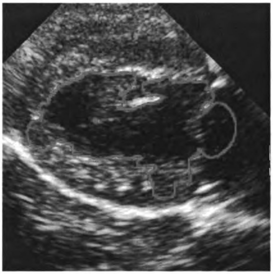


Figure 5.2: Endocardium detection in an echocardiogram (dog heart). Segmentation using the Geodesic Active Contour model. Note the ruggedness and the leakage through regions of low gradient, while stopping at extraneous anatomical structures inside the ventricle.

What *can* be done, however, if smooth boundaries are required in noisy images, is to search for minima of the energy (5.4) in a (finite dimensional) subspace of the space of all curves. One way to do this is to choose a finite set of functions $\{\mathcal{B}_i\}$ (with certain desirable properties), and consider only snakes which are linear combinations of those basis functions:

[8]Using the ITK [65] implementation of the geodesic active contour model.

$$\gamma_C(s) = \sum_{i=1}^{l} c_i \cdot \mathcal{B}_i(s) \tag{5.24}$$

The search space is reduced now from the *infinite dimensional* space of, say, continuous or, perhaps, twice differentiable curves ($\mathcal{C}^0$, respectively $\mathcal{C}^2$), to the *finite dimensional* space $span\langle \mathcal{B}_i \rangle \cong \mathbb{R}^{2l}$ and the restriction to this space of any snake energy function is simply a function of $2l$ variables. The shape of any $\gamma = \gamma_C$ is controlled by the vector $C = (c_1, \dots c_l) \in \mathbb{R}^{2l}$ of free parameters.

Several varieties of snakes fall in this framework. Staib and Duncan [100] express a closed curve in the form of a trigonometric series, and then, to reduce the number of degrees of freedom, they truncate the series to the first n harmonics, where n is a finite, user-defined integer. Along the same lines, Jain *et al.* [58] use products of the form $\sin nt \cdot \cos mt$ as basis functions. B-Splines [28] were used as active contours (B-Snakes) for border detection and tracking [5, 11, 93], and for stereo problems [74]. Cootes and Taylor [21] learn the modes of variation of the outline of an object of interest and express the snake as a linear combination of the most significant modes. Regarding the selection of the number of free parameters that define the snake, Figueredo [39] proposes a very powerful, fully automatic way of determining this number, using a minimum description length (MDL) criterion.

Parametric snakes are best suited for problems where smoothness is a must. In addition to that, they preserve topology, which, in medical images, is often known a priori. Furthermore, prior knowledge about the shape of the snake can be introduced in a more natural way, as prior probability on the coefficients, and can be done so in a way invariant to translations, rotations and scale [24] (although very recently, shape prior has been introduced in level sets too by Leventon *et al.* in [68], and Cremers *et al.* in [25]). Last, but not least, from a practical standpoint, minimization of an energy like (5.4) is a much more tractable problem in the case of parametric snakes:

such an energy is simply a function of $2l$ variables, and can be minimized by any of a number of well tested, already existing minimization algorithms. No special provisions must be made to ensure the numerical stability of the evolution problem.

In the next chapter we design a new "species" of parametric snakes which particularly targets extremely noisy images in which the target boundaries are known to be smooth.

Chapter 6

Interpolation Snakes

This chapter is central to our work, and represents one of the main contributions of the thesis. We introduce here a new kind of active contour based on *interpolation splines*, with smoothness and with shape memory built-in. The new snake is designed with the problem of border detection in ultrasound images in mind, but, as we shall see, it may be used in natural images, with varying backgrounds.

6.1 Interpolation snakes

We have argued that parametric snakes are likely to work better in difficult environments, such as ultrasound. Of these snakes, we shall single out one particular type, most suitable for imposing *spatial* hard constraints.

Virtually all snakes we have encountered in the literature are *closed* snakes. One reason for this is that objects have closed boundary, and usually one wants to detect the entire contour. However, with non-uniform boundaries, or with boundaries which are naturally piecewise smooth, it may be easier to detect several open segments of the boundary, and concatenate them, rather than detect the entire boundary at once. We therefore want to define *open* snakes, in addition to closed snakes. However, in the case of open snakes, the end points must be fixed throughout the fitting process, or else, the snake would shrink along the boundary, possibly to a single point. This is one reason why one might want to have hard spatial constraints on a snake.

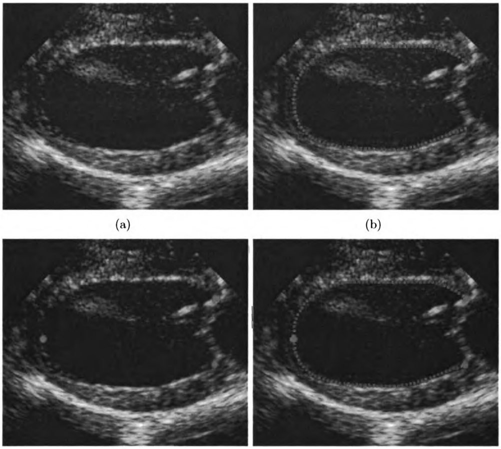
More importantly, there are situations where certain points or regions along the boundary of interest have particular significance for the problem at hand, and so, they have to be accurately located. Such is the case, for instance, with the two valve points and the apex of the left ventricle, in echocardiograms (See Figure 6.1 (a)). When these points happen to lie along weak edges or incomplete data, or simply in a region that does not differ qualitatively from other regions along the border, they are nearly impossible to detect by a "simple-minded" data fitting snake.

Short of implementing special purpose modules dedicated to the sole detection of these special points, a compromise solution might be to allow an expert user to mark these points as fixed at initialization time. These fixed points are hard constraints for an evolving active contour and we argue that among the various types of smooth snakes, *interpolation snakes* are the most amenable to imposing hard constraints. These were introduced in [77] and are one of the contributions of this thesis.

The snake *per se* is a disjoint union of one or more components, each of them modeled as a *cubic interpolation spline* (Section 2.3). Each spline is built using some 10 to 20 markers as control points, placed in the image by the user. See Figure 6.2. We parametrize the initial snake by arclength, as usual. What drives the snake is an energy potential defined as a function *on the control points.* We describe this in the following section.

6.2 The Energy Functional

Let $C = (y_0, \ldots, y_{l-1}) \in \mathbb{R}^{2l}$ be the control points of a spline $\gamma = (\gamma_0, \ldots, \gamma_{n-1})$. The breaks are as before (Section 2.3) $\xi = (\xi_0, \ldots, \xi_{l-1})$ and the degree is $k = 3$. The spline γ becomes a snake if we vary the parameters (in some coherent way, as to



(a) (b) (c) (d)

Figure 6.1: Spatial hard constraints on an active contour in cardiac ultrasound. These are the red points in (c) and (d). The two red points on the right are the *valve base points* and the single point on the left is the *apex*. The light matter between the two valve points is the actual valve (*mitral valve*), in a semi-open position. Since in any frame of the heart cycle the valve can be anywhere between fully open, to fully closed, it makes sense to detect only the actual heart wall (green curve), which is an open contour.

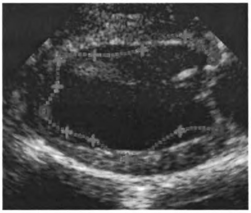


Figure 6.2: Interpolation snake. The control points are the red and blue markers. A blue circle (digit 0) is a fixed control point (hard constraint). A red plus sign, is a variable control point.

optimize some measure of fit). While it is possible to vary both the control points C as well as the breaks ξ, we shall only vary the control points; the breaks remain fixed. Since now γ is completely determined by its control points, we shall use the notation $\gamma = \gamma_C$.

We define the snake energy starting from Equation (5.4). Any energy functional $E = E(\gamma)$ defined on splines (or other parametric snakes), depends on the curve γ_C only via the vector of control parameters C, so our new energy will be a function

$$E = E(C) = E(y_0, \ldots, y_{l-1})$$

defined on $\mathbb{R}^{2l}$.

If any of the control points must remain fixed during the minimization (hard constraints), we have a situation in which the energy must only be defined on the subset of variable control points, while the spline must be computed on all the control points. Naturally, the basic idea is to skip over the fixed points when computing the energy. The implementation requires some care, but can be carried out in an elegant way, in C++, for instance, by overloading the index operator (*operator[]*) of the *vector* class.

Since smoothness is built into γ_C, we shall drop the internal energy term E_{int} from the energy (5.4). Our only concerns now are (a) to make the snake converge to boundaries, (b) keep the points equidistant during the evolution and, eventually, (c) impose a prior on the overall shape of the snake, e.g. in the form of a density function on the control vector C, or using kernel methods as in [24]. Under these requirements, we model the snake evolution by the following energy:

$$\begin{aligned} E(C) = & \underbrace{-\int_\gamma w(s) \cdot \langle \nabla I^\perp, \frac{\gamma'}{|\gamma'|} \rangle}_{E_{img}} \\ & + \underbrace{\int_0^L ||\gamma'(t)| - 1| dt}_{E_{arc}} \\ & + E_{shape} \end{aligned} \tag{6.1}$$

Here, $\langle \cdot, \cdot \rangle$ denotes the dot product, $\nabla I^\perp$ denotes a rotation by +90 degrees of ∇I, $C = (y_0, \dots y_{l-1}) \in \mathbf{R}^{2l}$ is again the vector of control points of the spline, $L = length(\gamma)$, and $w(s)$ is a user-defined weight function along γ.

We minimize the energy (6.1) using Powell's conjugate directions gradient descent algorithm [88].

6.2.1 Image energy

The **image energy** E_{img} measures the fit to data, and we define it as:

$$E_{img} = -\int_\gamma w(s) \cdot \langle \nabla I^\perp, \frac{\gamma'}{|\gamma'|} \rangle \tag{6.2}$$

Most commonly $w \equiv 1$, in which case, if we fix an orientation along γ, the spline will align itself along edges with dark to its right, and light to its left. The reason for this is that $-\langle \nabla I^{\perp}, \frac{\gamma'}{|\gamma'|}\rangle$ is smallest when $\nabla I^{\perp}$ is colinear with γ' and points in the *same* direction as γ'. This means that the gradient itself points from right to left as we walk along γ in the direction of the chosen orientation on γ. See Figure 6.3

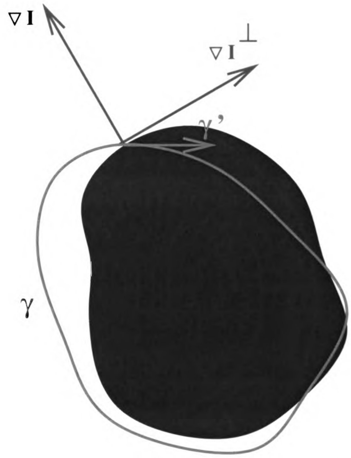


Figure 6.3: Image energy term at a single point: $E_{img} = -w \cdot \langle\ \nabla I^{\perp},\ \frac{\gamma'}{|\gamma'|}\rangle$. Red: active contour γ and tangent γ'. Black: target object. Blue: gradient ∇I and rotation by $+90^{\circ}$ (note that in image coordinates, positive angles are measured *clockwise*). If ∇I points to the *left* of γ, then $\langle\ \nabla I^{\perp},\ \dot{\gamma}\rangle > 0$, and the dot product is largest when $\nabla I \perp \dot{\gamma}$. Therefore, a weight $w > 0$ is needed in this case. Reversing the orientation on γ would require a weight $w < 0$.

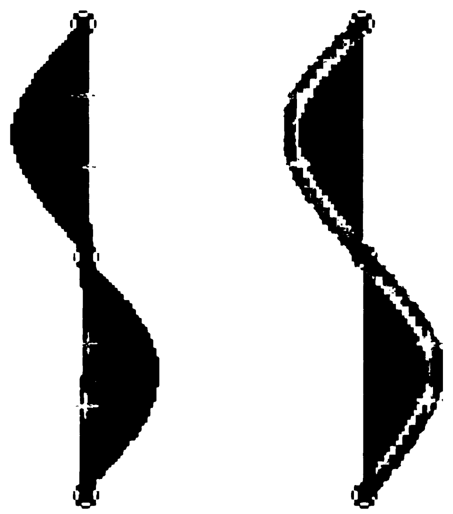
An illustrative example of the general case is shown in Figure 6.4. The green curves represent the initial (a) and final (b) interpolation snake. A blue circle represents a control point with $w = 0$ and will be kept fixed during the minimization. The variable points are marked in red, either with a plus sign ($w = +1$) or with a minus sign ($w = -1$). The energy (6.1) is a function of the red control points only. The weights at any other point on the spline (green points) are computed by interpolation of the known weights. In this example, the orientation of the spline is from top to bottom. Although initialized on an edge, the snake moves away from it, picking the edges with the polarity dictated by the weights: as we walk *downward* along the final snake (b), the $w < 0$ points are attracted by a *dark* $\rightarrow$ *light* edge, whereas points with $w > 0$ are attracted by a *light* $\rightarrow$ *dark* edge. This edge polarity selection by the snake is due to the fact that unlike the image energy E_{img} in (5.4) which only depends on the magnitude of the gradient, the E_{img} in (6.1) depends on the magnitude of the gradient *and* on the angle between the gradient ∇I and the (unit) tangent to γ. We have found that this latter image energy performs better than the original magnitude-only image energy.

A final, but important detail about E_{img} is the computation of the image gradient ∇I. Computation of the intensity gradient requires smoothing of the initial image, and the amount of smoothing determines the scale at which the snake evolution takes place. We compute the gradient in a way that allows us to seamlessly vary the scale, from coarse to fine, during the fitting process.

Instead of the usual Gaussian smoothing, or the more sophisticated anisotropic smoothing methods [86], we compute the directional derivatives at each pixel as

$$\partial_x I = \bar{I}_{right} - \bar{I}_{left}$$
$$\partial_y I = \bar{I}_{bottom} - \bar{I}_{top}$$

where each average intensity is calculated on a small rectangle, as shown in Figure 6.5.



(a) Initial (b) Final

Figure 6.4: Interpolation snake. Orientation along the snake is from top to bottom. Blue and red: control points. Green: arbitrary points on the snake. The weights of each of the control points are indicated by the shape (and color) of the marker: a blue circle means $w = 0$ (hard constraint), whereas a red $\pm$ means $w = \pm 1$. Despite starting out on an edge, the particular weights guide the snake towards other edges, as follows: $w < 0$ selects *dark* $\rightarrow$ *light* gradient, whereas $w > 0$ selects *light* $\rightarrow$ *dark* gradient, as we move along the curve with the chosen orientation. The points with $w = 0$ (blue circles) remain fixed during the fitting process, acting as hard constraints.

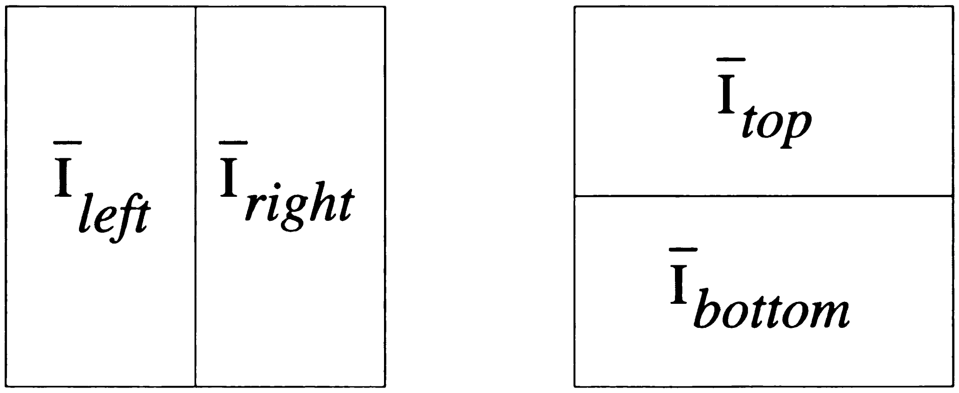


Figure 6.5: Computation of the image gradient.

Computation of the average intensity over a rectangle of *any* size can be carried out in *constant* time, regardless of the size of the rectangle, using only four look-up operations, by using a clever representation of the image I in the form of the *integral map* [107]:

$$\tilde{I}(x, y) = \int_0^x \int_0^y I(x, y) dx dy$$

Clearly (Figure 6.6), on any rectangle $[x_l, x_r] \times [y_l, y_r]$ we have

$$\int_{x_l}^{x_r} \int_{y_l}^{y_r} I(x, y) dx dy = \tilde{I}(x_r, y_r) - \tilde{I}(x_r, y_l) - \tilde{I}(x_l, y_r) + \tilde{I}(x_l, y_l) \qquad (6.3)$$

which makes possible the computation in constant time of the average intensity on any rectangle.

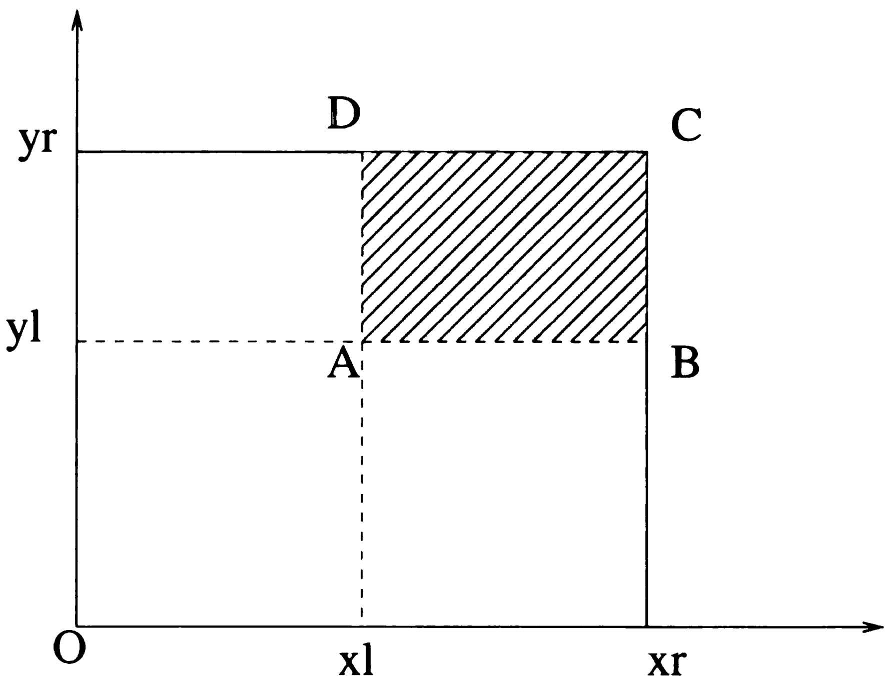


Figure 6.6: Integral map. The integral of I over the rectangle $ABCD$ can be computed by Equation (6.3), knowing the integrals of I over the four rectangles with the lower left corner at the origin O, and the upper left corner at A, B, C and D.

In the discrete setting, the integrals get replaced by finite sums. The integral map itself, can be computed in *quadratic* time using a straightforward, bottom-up dynamic programming algorithm, based on the following recurrence, obtained by taking $x_r = x_l + 1$ and $y_r = y_l + 1$ in (6.3):

$$\tilde{I}(x+1, y+1) = \tilde{I}(x, y+1) + \tilde{I}(x+1, y) - \tilde{I}(x, y) + I(x+1, y+1)$$

Note that this last relation gives $I(x+1, y+1)$ in terms of the values of $\tilde{I}$, so knowing the integral map of I we can recover exactly the image itself. In other words, the integral map is an invertible operator.

The size of the rectangular patches on which the gradient is computed determines the amount of smoothing, and hence, the amount of detail that will be used during the fitting: a larger scale ($\approx$ 30 × 60 rectangles), makes the snake sensitive to more distant edges, but produces a rather coarse fit, whereas a small scale (e.g., smaller than 10 × 20) makes the snake sensitive only to nearby edges, and produces a closer fit.

We illustrate *multi-scale* fitting of a *multiple-component* interpolation snake in Figure 6.7. The original image is noisy, with 20% salt-and-pepper noise. Figure 6.7 (b) shows the initial multi-component snake. There is a single (closed) component around the circle, and four *open* components around the square, concatenated. As before, the red "+" signs represent variable control pints with positive weight, and the blue circles represent fixed control points. Figure 6.7 (c) shows the detected boundary at a coarse scale (30 × 60 rectangles used in the computation of ∇I). The fitting continues at scales 20 × 40 and 10 × 20. The final result is shown in (d). Figures 6.7 (e) and (f) show the actual gradient (x-component only) used in (c) and (d), respectively.

Since computation of the integral map need only be done once, in one pass through the image, and since afterwards computation of the gradient ∇I can be done in

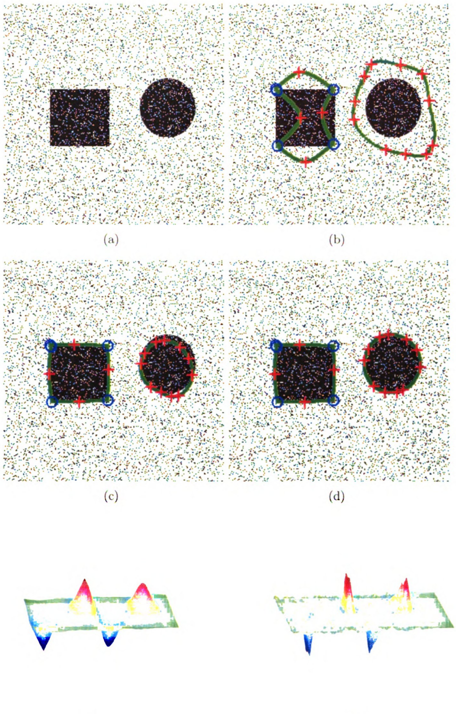

(a) (b) (c) (d) (e) (f)

Figure 6.7: Multi-scale, multiple-component interpolation snake. (a) 20% salt-and-pepper image; (b) Initial snake; (c) Coarse scale fit: 30×60 rectangles for computation of ∇I. (d) Fine scale fit: 10×20 rectangles. (e) Large scale gradient used in (c) (x direction only); (f) Small scale gradient used in (d) (x direction only);

constant time, regardless of scale, we can and do vary the scale during the fitting process, gradually, from coarse to fine.

6.2.2 Arclength energy

The **arclength energy** E_{arc} is designed to keep the points of the snake equidistant throughout the minimization process, by penalizing non-arclength parametrization.

$$E_{arc} = \int_0^L ||\gamma'(t)| - 1|dt \tag{6.4}$$

Since under the canonical parameter s, the speed along the curve is 1 (i.e. $|\gamma'(s)| = 1$), then the amount $\left||\gamma'(\tau)| - 1\right|$ is indicative to what extent τ fails to be the canonical parameter. The necessity of the arclength term is due to the following argument. Without exception, a curve γ is discretized by sampling it at equally spaced grid points $t_i = i \cdot \Delta t$. However, as we have seen in Section 2.2.1, the points $\gamma_i = \gamma(t_i)$ need not be equidistant, so this procedure does not correspond to the arclength parametrization. Of course, if a non-uniform distribution of the points along the snake is allowed, then the image energy will indicate a misleading measure of fit, if e.g., a densely populated segment along the snake happens to fall into an energy pit (strong edge). It is therefore crucial to start out with, and maintain equally spaced points on the snake.

This situation is illustrated in Figure 6.8, which shows a typical case. We fit an interpolation snake twice, starting from the same initial configuration: once using E_{img} only - hence without the arclength energy (6.4) - in Figure 6.8 (c), and one more time using both E_{img} and E_{arc} (Figure 6.8 (e)). For visual comparison against the ground truth (Figure 6.8 (a)), we overlay each result on top of the ground truth (Figures 6.8 (d) and (f), respectively). In the absence of the arclength energy, the goodness of fit is measured by the image energy alone, and has a value of -44.7159, "better" even than the image energy along the ground truth itself (-16.4465), erroneously indicating a very good fit. When the arclength energy is combined with the image energy, the fit is visibly better than in the former case. Quantitatively, in each case we measure the difference between the detected boundary and the ground truth using e.g., the modified (average) Hausdorff distance [32]. Without the arclength energy, this measure is 7.68 pixels, whereas with the arclength energy, the average Hausdorff distance is only 3.60 pixels.

Somewhat surprisingly, the problem of preserving arclength parametrization along the snake during the evolution, although noticeable both theoretically *and* practically, has been silently overseen, and left as an implementation detail. Different active contours implementations deal with this problem in an ad-hoc manner, e.g. by removing points from regions along the snake where the points have come too close together, and inserting points in regions where the snake has become too sparse. The first authors to address this problem were, to our knowledge, Williams and Shah [110], who replace the first order term in the internal energy E_{int} (5.3), by a term which penalizes for deviations from the mean distance $\bar{d} = \sum|\gamma_{i+1} - \gamma_i|/n$ along the contour, namely $\sum(|\gamma_{i+1} - \gamma_i| - \bar{d})$.

In the discrete setting, we take an *equidistant* subdivision of the interval $[0, L]$, namely $\tau_i = i \cdot L/n$, and let $\gamma_i = \gamma_C(\tau_i)$. We approximate the derivative at τ_i as $\gamma'(\tau_i) \approx (\gamma_{i+1} - \gamma_i)/\Delta\tau$, and approximate the arclength energy E_{arc} by a Riemann sum:

$$E_{arc} \approx \sum_{i=0}^{n-1} \left| \frac{|\gamma_{i+1} - \gamma_i|}{\Delta\tau} - 1 \right| \cdot \Delta\tau \tag{6.5}$$

$$= \sum_{i=0}^{n-1} \Big| |\gamma_{i+1} - \gamma_i| - \Delta\tau \Big| \tag{6.6}$$

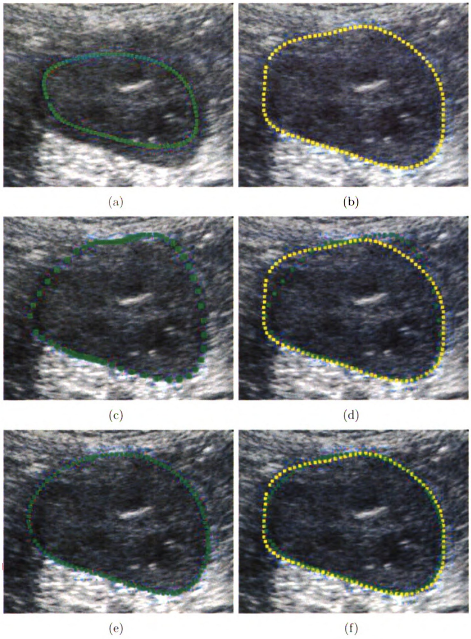


Figure 6.8: The need for arclength parametrization. Green: interpolation snake (control points omitted). Yellow: ground truth. (a) Initial snake (arclength parametrization). (b) Manually drawn border of a human kidney. (c) Snake fitted using image energy only. The points do not remain equidistant, and tend to accumulate along strong boundary segments, producing an artificially good score ($E = E_{img} = -44.7159$), better even than the ground truth score ($E = E_{img} = -16.4465$), but a poor fit. Average Hausdorff distance from ground truth: 7.68 pixels. (d) Same as (c), overlaid over the ground truth. (e) Snake fitted using both the image energy E_{img} and the arclength energy E_{arc}. Average Hausdorff distance from ground truth: 3.60 pixels. (f) Same as (e), overlaid over the ground truth.

If L is chosen to approximate the length of the (initial) contour, and if this length remains roughly the same during the evolution, then

$$\sum_{i=0}^{n-1} |\gamma_{i+1} - \gamma_i| \approx L = n \cdot \Delta\tau$$

so $\Delta\tau$ is the mean of all distances between consecutive points along the snake. We arrive this way at the elastic energy proposed by Williams and Shah [110].

The initial snake is also constructed so that the sampling at uniform time intervals corresponds to (an approximate) arclength parametrization. This was explained in Section 2.3

6.2.3 Shape energy

The **shape energy** E_{shape} represents the shape prior knowledge and can be imposed e.g., as a distribution on the vector of coefficients. Any of a number of techniques for distribution estimation can be used, such as e.g., Principal Component Analysis (PCA) on a number of training contours, Parzen windows, or more general kernel methods [24, 25]. In our experiments we use the *average chamfer transform* model, discussed in detail in Chapter 4.

Specifically, let $\gamma_1, \dots \gamma_k$ be *registered* training samples, represented respectively by their chamfer transforms $C_1, \dots C_k$. Let

$$\bar{C} = \frac{1}{k} \sum C_i$$

the average chamfer transform, and let μ be the curve that minimizes it:

$$\mu = \arg\min_{\mu'} \bar{C}(\mu') \qquad (6.7)$$

If γ_C is an interpolation snake, then we let T_C be the similarity transform that does *semiaxes alignment* (Section 3.4) of γ_C onto μ, define the shape energy as

$$E_{shape}(C) = s(\bar{C}(T_C(\gamma_C))) \tag{6.8}$$

where the function $s = s(x)$ has the form

$$s(x) = a^2 \cdot \frac{x^2}{x^2 + b^2}$$

for some constants a and b. The reason for "squashing" the average chamfer transform by composing with s is to bring $\bar{C}$ within a fixed, controlled range (Figure 6.9).

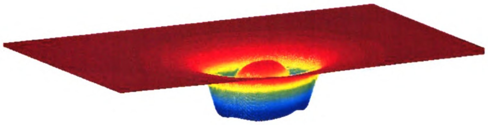


Figure 6.9: Squashed average chamfer transform, used as the shape model with interpolation snakes. Compare to Figure 4.3

Note that even though the shape energy is defined as a function on the control vector C, at each iteration we register the entire spline γ_C to μ (not just the control points!). The actual similarity to the training samples is computed again on the entire γ_C. Both the registration of γ_C to μ and evaluation of $\bar{C}(T_C(\gamma_C))$ have time complexity linear in the number of points on γ_C, the same as the complexity of the other two energy terms E_{img} and E_{arc}.

The benefits and limitations of the shape energy should be well understood and not overestimated: the shape energy, in general, can correct for significant deviations from the general shape of the training samples, but cannot account for details in any particular test case.

The example in Figure 6.10 illustrates the the benefits of using the shape energy. A clear kidney ultrasound image (Figure 6.10 (a)) is occluded by applying a white stripe with dark islands (Figure 6.10 (b)), in an attempt to "confuse" the snake. Figures 6.10 (c) and (d) show the detected boundary *without* the shape energy term (c), and *with* E_{shape} (d). The "distractor" stripe tends to deform the snake toward a shape not part of the model, and the shape energy term manages to correct for that deviation.

What the shape energy *cannot* do, is to correct the shape of a snake (towards the desired boundaries), if the snake already happens to be similar to the training shapes. We illustrate this point in Figure 6.11. Starting from the same initial contour (Figure 6.11 (a)), we fit the snake *with* (Figures 6.11 (c) and (d)) and *without* shape energy (Figures 6.11 (e) and (f)). The results in both cases are similar, and similar to the training shapes (on average), yet different from the ground truth (yellow). The reason is that unlike in the previous example, there is no distractor to alter the shape of the snake *significantly*, from the learned model. The shape energy has nothing to correct for, and has a minimal effect, because near convergence, the snake is already from the distribution of the training samples. In fact, without the shape energy, the average Hausdorff distance between the detected boundary and the ground truth was 5.13 pixels, whereas with the shape energy, the average Hausdorff distance was slightly higher: 5.85 pixels.

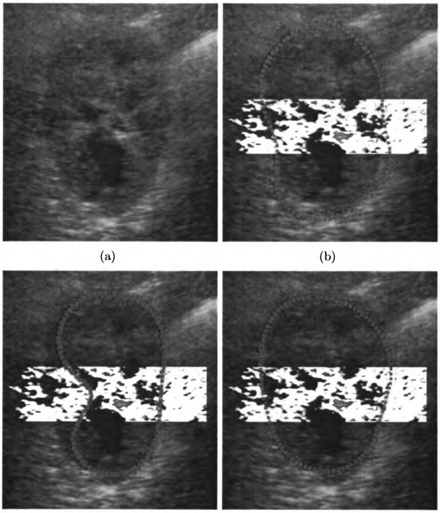

(a) (b) (c) (d)

Figure 6.10: Effect of the shape energy E_{shape}. (a) Clear kidney image. (b) Occluded kidney, and initial snake (green). (c) Final snake with no shape energy. (d) Final snake with shape energy.

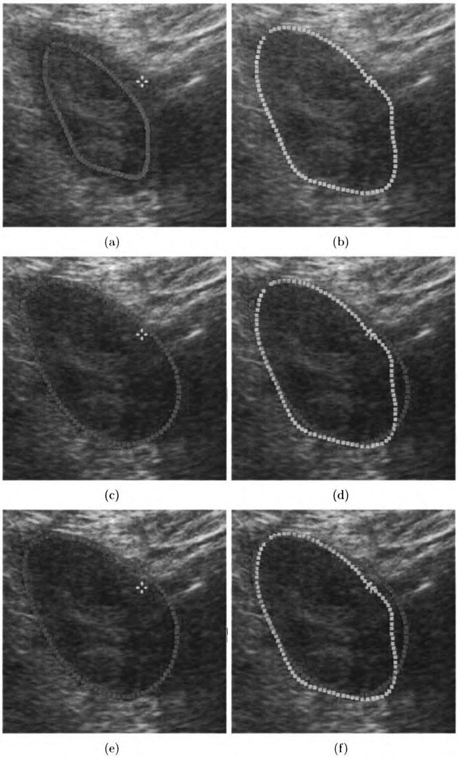


Figure 6.11: Limitations of the shape energy. (a) Initial snake. (b) Ground truth. (c) Final snake *without* shape energy. (d) Comparison of (c) with the ground truth. (e) Final snake *with* shape energy. (f) Comparison of (e) with the ground truth.

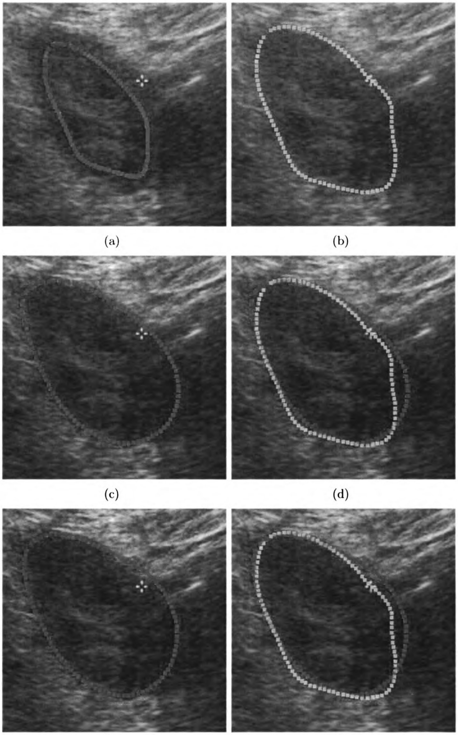


Figure 6.11: Limitations of the shape energy. (a) Initial snake. (b) Ground truth. (c) Final snake *without* shape energy. (d) Comparison of (c) with the ground truth. (e) Final snake *with* shape energy. (f) Comparison of (e) with the ground truth.

6.3 Conclusion to Chapter 6

We have defined here a type of active contour - *interpolation snakes* - with shape memory, smooth by construction, and that allows for hard spatial constraints. This is achieved by means of an *interpolation* spline, preferred over other splines because the spline passes through all its control points.

The snake can have multiple connected components (Figure 6.7), and each component preserves topology during the boundary finding process. This is not a drawback in applications such as medical imaging, where the topology of the object of interest is generally known a priori, it is almost a "must-have" feature.

A three-term energy functional is being minimized (Equation (6.1)). The main term (E_{img}, Equation (6.2)) is the boundary-seeking component. Unlike the image term in the original snakes algorithm [63], which only seeks pixels with large intensity gradient, our image energy term guides the snake based on the angle between the tangent to the snake and the image gradient (Figure 6.3). In particular, this makes the snake sensitive to the edge orientation[1] (Figure 6.4). The second term (E_{arc}, Equation (6.4)) helps the snake remain parametrized by arclength, thus preventing the constituent points of the snake to accumulate near very strong edges in the case of objects with non-uniform boundaries (Figure 6.8). Last, but not least, there is a shape prior term (E_{shape}, Equation (6.8)) that prevents significant deviations of the snake from a known, predefined shape model. The three terms must be commensurate, so some normalization is in order.

The choice of splines for the active contour reduces the search space from an infinite dimensional space of $\mathbf{C}^2$ curves in $\mathbb{R}^2$ to a low dimensional $\mathbb{R}^{2k}$, where k is the number of control points. As such, the energy E is a function of $2k$ independent variables and can be minimized by standard routines, some of which do not require

[1]This feature can be turned off, however, by taking absolutee value in the definition of E_{img} (Equation (6.2))

computation of the gradient, thus avoiding computation of the Euler-Lagrange equations.

Chapter 7

A Different Direction: Break Point Detection

We devote this chapter to a non-variational approach to boundary detection. The new approach finds the boundary by detecting jumps in the intensity profile along segments across the boundary of the object of interest. The core of this approach is a general, mathematically sound algorithm for detection of jumps in a 1D signal. We expect this algorithm to have applications well beyond border detection. It is one of the contributions of the thesis.

7.1 One-Dimensional Break Detection

We arrived at the problem of break (or jump) detection in 1D signals while trying to find the kidney boundaries in ultrasound images. By and large the kidney has overall lower intensity than its surroundings. Thus, if AB is a segment that goes across the boundary of the kidney, for any point $P \in AB$ we can compute the average intensity on AP and on PB. Intuitively, the border point we seek should be the one for which one average is as low as possible while the other is as high as possible. See Figure 7.1 and also Example 7.1.2.

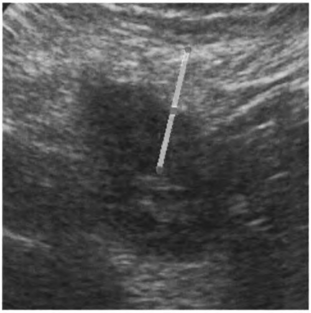


Figure 7.1: Boundary detection by finding a break point in an intensity profile across the boundary.

It is a trivial matter to detect a *single* break point: simply move along the profile and choose as break point, the point which maximizes the difference between the left and right average intensity. From the very beginning, it is clear that the problem of break point detection should be formulated in the more general case, that of *multiple* break points. It should also be clear that we cannot expect a break point detection algorithm to work well for arbitrary 1D signals: in this motivational example the intensity on either side of the break point is *relatively constant.* After some thought, we have come up with the right mathematical model of the situation we just described. We present this model next.

For any interval $[a, b] \subset \mathbb{R}$, let the set of *n-segment partitions of* $[a, b]$ be

$$\mathcal{P}_n = \{T = (a = t_0 < t_1 \cdots < t_n = b)\}$$

Let $H = (h_1, \dots, h_n) \in \mathbb{R}^n$ be any vector. Each $T \in \mathcal{P}_n$ and $H \in \mathbb{R}^n$ define a piecewise constant[1] function:

$$y_{H,T} = \sum_{i=1}^{n} h_i \cdot \chi_{[t_{i-1},t_i)} \tag{7.1}$$

where for any interval I, χ_I is the characteristic function of I. Since the components of H appear as coefficients in Equation (7.1), we shall call the vector H the *weights vector*. Denote by $\mathcal{PC}$ the space of *piecewise constant functions* on $[a, b]$:

$$\mathcal{PC} = \{y_{H,T} | H \in \mathbb{R}^n, T \in \mathcal{P}_n\}$$

Let now y be a noisy version of $y_{H,T}$:

$$y = y_{H,T} + \eta \tag{7.2}$$

In practice, we can only measure the noisy signal y (H and T are unknown), and we would like to *segment* it, assuming some information about the noise η. In the most general setting, this means that we want to find

1. the number n of partition segments,

2. the actual partition $T = (a = t_0 < t_1 \dots t_n = b)$,

3. the weights vector $H = (h_1, \dots, h_n)$.

Segmentation of y is impossible without assumptions on the noise function η. For instance, if $\eta = -y_{H,T}$, then $y = 0$, from which nothing can be inferred. We shall impose conditions on the noise η which will guarantee mathematically that $y_{H,T}$ is closest in L^2 norm to y, among all piecewise constant functions. That is, if for each piecewise constant function $y_{G,S} \in \mathcal{PC}$ we define the L^2 energy

[1]In Lebesgue Theory, these are called *simple functions*. We find the term *piecewise constant* more suggestive in the context of segmentation.

$$E(y_{G,S}) = \int_a^b |y - y_{G,S}|^2 dt \tag{7.3}$$

then we want $y_{H,T}$ to be its minimum. The following proposition asserts under which conditions of noise this is true.

Proposition 7.1.1 *If $y = y_{H,T} + \eta$ and $\int_u^v \eta(t) = 0$ on any subinterval $[u, v] \subset [a, b]$, then*

$$E(y_{G,S}) \geq E(y_{H,T})$$

for any piecewise constant $y_{G,S}$.

Proof:

$$\begin{aligned}
E(y_{G,S}) &= \int_a^b |y - y_{G,S}|^2 dt \\
&= \sum_{i=1}^{n} \int_{t_{i-1}}^{t_i} |h_i - y_{G,S}(t) + \eta(t)|^2 dt \\
&= \sum_{i=1}^{n} \left(\int_{t_{i-1}}^{t_i} |h_i - y_{G,S}(t)|^2 dt + 2 \int_{t_{i-1}}^{t_i} (h_i - y_{G,S}(t)) \cdot \eta(t) dt + \int_{t_{i-1}}^{t_i} \eta^2(t) dt \right) \\
&= E(y_{H,T}) + 2 \sum_{i=0}^{n} \int_{t_{i-1}}^{t_i} (h_i - y_{G,S}(t)) \cdot \eta(t) dt + \int_a^b \eta^2(t) dt \\
&= E(y_{H,T}) + \int_a^b \eta^2(t) dt
\end{aligned}$$

The last equality is true because $h_i - y_{G,S}$ is piecewise constant on $[t_{i-1}, t_i)$, and because by hypothesis, η integrates to 0 on any subinterval.

□

Remark 7.1.1 *While Proposition 7.1.1 is undoubtedly true, unfortunately there are not many functions η besides the zero function which satisfy the hypothesis $\int_u^v \eta =$*

0 for any subinterval $[u, v] \subset [a, b]$. To be integrable η would have to be Lebesgue measurable, but a result from real analysis asserts that if η integrates to zero on arbitrarily small intervals around a point, then the function must be zero at that point [92]. If we require that η integrate to zero on any subinterval, then η must be zero at all points (except possibly for a set of measure zero).

However, there is a fix to this problem, inspired again from practice. If we choose some scalar $\lambda > 0$ and only require that η integrate to zero only on subintervals of length no less than λ, then we can find many such functions. Divide up the interval $[a, b]$ into a large number of equal length subintervals, and call their common, nonzero length λ. On each of such subintervals consider arbitrary Haar-like functions, i.e., functions symmetric with respect to the midpoint of the interval. Because of symmetry, any such function integrates to zero on its respective subinterval. If η is now any combination of such functions, then η integrates to zero on any union of length-λ subintervals. Accordingly, we can no longer work with arbitrary partitions of the whole interval $[a, b]$, but only with partitions whose break points are multiples of λ. With these modifications, the proof of Proposition 7.1.1 remains unchanged.

This modification amounts precisely to the discretization of the problem. Thus, in summary, Proposition 7.1.1 is true in the continuous case, but not interesting, as the noise η must be zero to begin with. Discretization allows for non-trivial noise, and, with the proper modifications the result remains true. For clarity and simplicity, however, we state the results and present the proofs in the continuous framework.

In segmentation problems or 2D boundary detection problems one defines an energy functional heuristically, in such a way that the e.g., the boundaries of interest are expected to be among the *local* minima of that energy functional. Usually there is no proof that the minima of the energy agree with the human perception, and it is often difficult to control which minima the minimization procedure is going to find. By contrast, Proposition 7.1.1 asserts that if $y = y_{H,T} + \eta$ is a noisy signal, then its defining piecewise constant core $y_{H,T}$ - which we would like to recover - is the *absolutee* minimum of the L^2 energy (7.3), so by minimizing the energy (7.3) after all $G \in \mathbb{R}^n$ and $S \in \mathcal{P}_n$, we recover the weights H and the partition T.

We can do even better though. If we assume the noise η has zero mean on any subinterval of $[a,b]$[2], then the weight h_i is precisely the mean of the signal y on $[t_{i-1}, t_i)$. In other words, the weights H are not independent on the partition T, but rather they are uniquely determined by T (and by the signal y itself). This suggests that we can consider the energy E only as a function of the partition (rather than partition *and* weights). To prove this fact, let us first introduce some notation.

Let $S = (a = s_0 < s_1 < \cdots < s_n = b)$ be a partition of $[a,b]$. Let $Y_S = (\bar{y}_1, \ldots \bar{y}_n)$ be the weight vector whose components $\bar{y}_i$ are the mean of y on $[s_{i-1}, s_i)$.

Proposition 7.1.2

$$E(G,S) \geq E(Y_S, S) \qquad \forall G \in \mathbb{R}^n$$

Proof: :

$$\begin{aligned} E(G,S) &= \int_a^b |y - y_{G,S}|^2 \\ &= \sum_{j=1}^n \int_{s_{j-1}}^{s_j} (y - \bar{y}_j + \bar{y}_j - g_j)^2 dt \\ &= \sum_{j=1}^n \Big(\int_{s_{j-1}}^{s_j} (y - \bar{y}_j)^2 dt + 2(\bar{y}_j - g_j) \cdot \underbrace{\int_{s_{j-1}}^{s_j} (y - \bar{y}_j) dt}_{0} + \int_{s_{j-1}}^{s_j} (\bar{y}_j - g_j)^2 dt \Big) \\ &= E(Y_S, S) + |y_{Y_S,S} - y_{G,S}|_{L^2} \end{aligned}$$

where we have used that $\bar{y}_j = \int_{s_{j-1}}^{s_j} y(t)dt/(s_j - s_{j-1})$ is the mean of y over

[2]Again, in the sense of Remark 7.1.1.

$[s_{j-1}, s_j)$.

□

It follows from this proposition, that in order to segment the signal y, it suffices to minimize the energy $S \longrightarrow E(Y_S, S)$, rather than $(G, S) \longrightarrow E(G, S)$. To summarize,

Corollary 7.1.1 *Suppose $y = y_{H,T} + \eta$ with $\int_u^v \eta dt = 0$ for any subinterval $[u, v] \subset [a, b]$. For any partition $S \in \mathcal{P}_n$, let y_j be the mean of y on $[s_{j-1}, s_j)$, and let $Y_S = (\bar{y}_1, \dots, \bar{y}_n)$ be the weight vector of the means and $\bar{y}_S = y_{Y_S, S}$ (the piecewise constant function defined by Y_S and S). Define the energy*

$$E(S) = \int_a^b (y - \bar{y}_S)^2 \tag{7.4}$$

Then

$$T = \arg \min_{S \in \mathcal{P}_n} E(S), \qquad H = Y_T$$

That is, the partition T is the absolutee minimum of E, and the weights vector H consists of the means of y on each of the segments of T.

Certainly, the problem now is to minimize the energy (7.4) with respect to $S \in \mathcal{P}_n$. Equivalently, we must minimize E after $S \in \mathcal{R}^n$, *subject to the constraints* $a = s_0 < s_1 < \cdots < s_n = b$. In the discrete case, if y is given as a sequence $(y_0, \dots, y_N)$, then for large N and n an exhaustive search for the optimal T would have complexity $O(N^n)$ ("N-choose-n"). Fortunately, a very elegant *dynamic programming* solution exists, based on the following observation (stated here in the continuous case):

Proposition 7.1.3 *Suppose $T = (a = t_0 < t_1 < \cdots < t_{n-1} = b < t_n = c)$ is the optimal partition on $[a, c]$ (i.e., T minimizes E (Equation (7.4)) over $[a, c]$). Let $T' = (a = t_0 < t_1 < \cdots < t_{n-1} = b)$ be the truncation of T up to the last interior break point $b = t_{n-1}$. Then T' is the optimal partition over the subinterval $[a, b] \subset [a, c]$.*

Proof: Since we're working with different intervals, we must specify the dependence of the energy E on the interval and on the number of segments. Thus $E_{a,c,n}$ is the energy for partitions over $[a,c]$ with n segments, and $E_{a,b,n-1}$ is the energy for partitions over $[a,b]$ with $n-1$ segments. Here $b = t_{n-1}$. As a matter of notation, for any interval $[u,v]$ let $\bar{y}_{[u,v]}$ denote the mean of y over $[u,v]$. Obviously, by the very definition of E (Equation (7.4)), we have

$$E_{a,c,n} = E_{a,b,n-1} + \int_b^c (y - \bar{y}_{[b,c]}))^2 dt$$

Let $S' = (a = s_0 < s_1 < \cdots < s_n - 1 = b) \in \mathcal{P}_{n-1}$ be the optimal partition over $[a,b]$. Then the partition $S = (S', c)$ has n segments, and spans $[a,c]$. Since among such partitions T is optimal, we have

$$E_{a,c,n}(T) \leq E_{a,c,n}(S)$$

and by subtracting $\int_b^c (y - \bar{y}_{[b,c]})^2$ on both sides we get

$$E_{a,b,n-1}(T') \leq E_{a,b,n-1}(S')$$

But since S' is optimal on $[a,b]$, we also have the opposite inequality:

$$E_{a,b,n-1}(T') \geq E_{a,b,n-1}(S')$$

□

Corollary 7.1.2 *Let $E^*_{a,c,n}$ be the minimum of $E_{a,c,n}$, and $E^*_{a,b,n-1}$ be the minimum of $E_{a,b,n-1}$, for any $a < b < c$. Then*

$$E^*_{a,c,n} = \min_{b \in [a,c]} \left(E^*_{a,b,n-1} + \int_b^c (y - \bar{y}_{[b,c]})^2 \right) \tag{7.5}$$

Once we have established the recurrence relation (7.5), the bottom-up dynamic programming algorithm is straightforward: given a one-dimensional signal $y = ((y_0, \ldots, y_N)$ *and* a desired number of segments n, we fill incrementally a matrix $E^*(b, k)$ for integers $0 \leq b \leq N$ and $0 \leq k \leq n$ using Equation (7.5). The value $E^*(N, n)$ is the minimum energy for partitions over the entire interval $[0, N]$ with n segments. The actual optimal partition T is then obtained by backtracking the indices b which achieve the minimum on the right-hand-side of (7.5).

Example 7.1.1 *For this example refer to Figure 7.2. We generate a random 4-segment integer partition T of the interval* $[0, 1000]$

$$T = (0, 169, 877, 924, 1000)$$

and a random weight vector

$$H = (7.17897, 96.6553, -66.3123, -24.9187)$$

The piecewise function $y_{H,T}$ determined by H and T is shown in red. This is the signal before adding noise, and can be seen as ground truth. Next, we construct the noisy signal

$$y = y_{H,T} + \eta$$

where η is Gaussian noise with zero mean and standard deviation $\sigma = 20$. The resulting noisy signal is shown in blue.

The result of the dynamic programming segmentation algorithm is shown in green. The detected partition T' and weights vector H' are

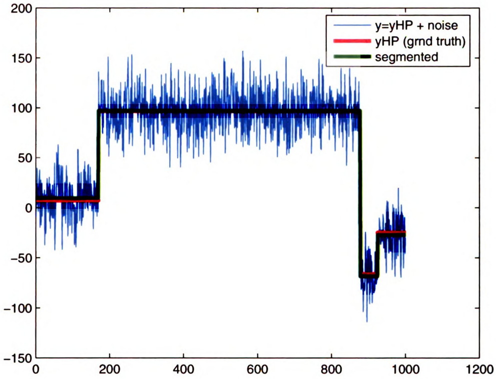


Figure 7.2: Break point detection of a randomly generated signal. Blue: noisy data $y = y_{HT} + \eta$ with 1000 points and 4 segments. Red: piecewise constant function $y_{H,T}$ used to generate y (ground truth). Green: denoised signal. The noise η was Gaussian with zero mean and standard deviation $\sigma = 20$. The true break points were $T = (0, 169, 877, 924, 1000)$ and the true weights vector was $H = (7.17897, 96.6553, -66.3123, -24.9187)$. The denoised signal has breaks *identical* to the true breaks, and weights $H' = (9.41748, 97.0672, -68.4484, -27.1419)$.

$$T' = (0, 169, 877, 924, 1000) \tag{7.6}$$

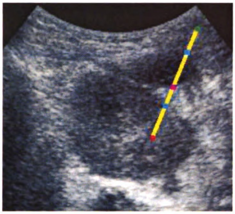
$$H' = (9.41748, 97.0672, -68.4484, -27.1419) \tag{7.7}$$

The detected breaks are identical to the ground truth breaks, and the detected weights are close to the original weights. This should not come as a surprise, as the mathematics leading to the dynamic programming algorithm guarantees *that the algorithm finds the ground truth $y_{H,T}$.*

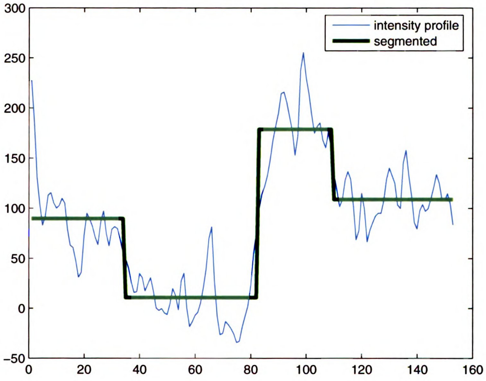
Example 7.1.2 *In this example we extract an intensity profile along a segment in an ultrasound image, and apply the break point detection algorithm. Refer to Figure 7.3. The the intensity profile is extracted along the yellow segment in Figure 7.3(a), moving from the top green end towards the red end. This profile is plotted in blue in Figure 7.3 (b). The break detection algorithm is applied, and the resulting piecewise constant approximation of the profile is plotted in (b) (green). The interior break points are marked along the yellow segment in (a) with cyan (for a high to low break) and with magenta (low to high).*

Aside from these toy examples, we shall present a real application of this algorithm in Chapter 8.5. In that application, we detect break points in ultrasound images along rays of equal length, originating from a point, for the purpose of border detection.

In fact, although we presented the theory first, the road-map was from concrete to abstract. We were looking for single break points along radial intensity profiles, optimal with respect to some metric. It turned out that the energy (7.4) was giving good results, and for two-segment partitions, the break point was readily found by a linear scan of the profile. Since the results were encouraging, we devised the experiment in Example 7.1.1, in which the ground truth is known: starting with a known piecewise constant function $y_{H,T}$, segment a noisy version of it $y = y_{H,T} + \eta$. In



(a)



(b)

Figure 7.3: Break point detection along an intensity profile in an ultrasound image. (a) We move along the yellow segment from the top green end towards the red end. The detected interior points are marked with cyan (high to low) and magenta (low to high). (b) Blue: 1D intensity profile along the yellow segment from (a). Green: segmented signal.

all experiments, the denoised signal was surprisingly close (almost identical) to the ground truth, and that could not be by chance: it had to be a mathematical explanation for it. We were able to provide rigorous proof to support the experimental findings. Moreover, the mathematics showed that there was nothing special about two-segment partitions, as we originally started out with, and that the results were in fact valid for partitions with any number of segments. Last, but not least, we were able to find a dynamic programming algorithm that would detect these partitions.

The only limitation of this work is that the number of partitions has to be specified a priori. It would be highly desirable to be able to choose the number of segments automatically, but we currently do not have a solution for that. We contend, however, that an MDL[3] solution to this problem is very likely.

[3]Minimum Description Length.

Part III

Applications

Chapter 8

Experimental Results

In this chapter we present some experiments designed to test the performance of the interpolation snakes introduced in Chapter 6. We do a systematic and objective study of four active contours models, including the interpolation snakes on synthetic noisy images (Section 8.4) and on kidney ultrasound images (Sections 8.2.2, 8.2.1, 8.2.3 and 8.2.4) by measuring the shape similarity of the detected boundaries against manually traced ground truth. For clinical validity of our algorithm we perform also a subjective blind study (Section 8.3), in which a human expert chooses the better of of two algorithms displayed simultaneously in separate copies of the same image.

In addition to the results on ultrasound, we demonstrate in Section 8.6 good performance of interpolation snakes for shape detection in non-medical images. We follow up the 1D break point detection algorithm from Chapter 7 with some partial results in Section 8.5.

8.1 Introduction

We have developed, so far, a framework for boundary detection, specifically targeting ultrasound images. In this chapter of the thesis we assess the performance of our algorithms. We do this objectively and systematically, by comparing the boundaries detected using our methods against the true boundaries (ground truth). In the case of real images (such as ultrasound), the ground truth usually consists of manually traced contours by human experts, while for synthetic images, the ground truth is the curve used to generate the synthetic image, i.e., the curve whose interior is the object of interest. We describe the types of images used in Section 8.1.3.

8.1.1 Performance Measures

One way to evaluate the performance of active contours is to compare them against ground truth curves. We use three measures to compare the similarity of two curves. Let X and Y be two simple closed curves in the plane.

Hausdorff Distance. This has been defined in Section 3.1 (3.6) as

$$H(X,Y) = \sup\{d(X,Y), d(Y,X)\}$$

where $d(X,Y) = \sup_{x \in X} d(x,Y)$ is the directed distance between the sets X and Y. Here, X and Y can be arbitrary sets in the plane, not necessarily curves.

The measure satisfies the mathematical definition of a distance, i.e., it has the following properties [47]:

1. $H(X,Y) \geq 0$, with equality if and only if $X = Y$.
2. $H(X,Y) = H(Y,X)$ (symmetry property).
3. $H(X,Z) \leq H(X,Y) + H(Y,Z)$ (triangle inequality).

This measure is known to be highly sensitive to outliers. To exemplify, suppose the two sets X and Y are identical, except for one point p. That is, $Y = X \cup \{p\}$. Then obviously $d(X,Y) = 0$ as $X \subset Y$, but $d(Y,X) = d(p,X)$, so the Hausdorff distance is $H(X,Y) = d(p,X)$. As such, the Hausdorff distance can be made arbitrarily large, by moving *a single point* towards infinity, even though the two sets may otherwise be identical. In image processing, when comparing two figures (e.g., each as the result of an edge detector) using the Hausdorff measure, the presence of outliers may deem the two figures as different, even though they may in fact represent the same object.

One way to make the Hausdorff distance less sensitive to outliers is to replace the *supremum* with the *average* when computing the directed distance $d(X, Y)$. This leads to the following similarity measure (introduced in [32] as the "modified Hausdorff distance").

Average Hausdorff Distance. This too has been defined in Section (3.8):

$$H_{av}(X,Y) = \sup\{d_{av}(X,Y), d_{av}(Y,X)\}$$

where $d_{av}(X,Y) = \int_X d(x,Y)dx/|X|$ is the directed average distance. As in the case of the Hausdorff distance, X and Y can be arbitrary sets. Here $|X|$ denotes the length of the curve X. It has been observed [32] that this measure does not satisfy the triangle inequality , so it is not a distance in the mathematical sense (properties 1,2 and 3 defined above).

Normalized Symmetric Difference.

$$NSD(X,Y) = \frac{|(X \setminus Y) \cup (Y \setminus X)|}{|X \cup Y|} \tag{8.1}$$

This is a measure that has not been mentioned in this thesis so far.It measures the amount of overlap between the sets X and Y. Strictly speaking, if X and Y are curves, on the right-hand-side of (8.1) we have the *interiors* of X and Y, respectively, but for simplicity, we denote these by the same letters as their boundaries. $|X|$ stands for the area of the set (enclosed by) X.

The numerator is the usual symmetric difference, and the denominator is a normalization factor. Obviously, $0 \leq NSD(X,Y) \leq 1$ for any sets X and Y, and $NSD(X,Y) = 0$ if and only if $X = Y$. $NSD(X,Y) = 1$ if and only if X and Y are disjoint.

Claim: The normalized symmetric difference does not satisfy the triangle inequality.

Proof: To prove the claim it suffices to find a counterexample. Let X and Y be arbitrary sets with non-empty intersection: $X \cap Y \neq \Phi$ and such that neither set is included in the other: $X \not\subseteq Y$ and $Y \not\subseteq X$. The triangle inequality for X, $\underbrace{X \setminus Y}_{Z}$ and Y reads

$$NSD(X,Y) \leq NSD(X, X \setminus Y) + \underbrace{NSD(X \setminus Y, Y)}_{0}$$

$$\frac{|X \setminus Y| + |Y \setminus X|}{|X| + |Y \setminus X|} \leq \frac{|X \cap Y|}{|X|}$$

Now if we keep X unchanged and expand Y so that $|Y \setminus X| \to \infty$ while maintaining $X \cap Y$ constant, then the left hand side in the last inequality approaches 1 asymptotically, so it will exceed the right hand side, which remains fixed, strictly less than 1.

Observe that none of these metrics measures *shape* similarity per se. For instance, suppose the two sets X and Y are two circles of the same radius. These two figures definitely have the same shape, so a true shape similarity measure should reflect that. However, all of the above measures are 0 on X and Y if and only if the two circles are concentric. If the two circles are disjoint, say, like the wheels of a bicycle, then $NSD(X,Y) = 1$ (largest possible), whereas the two Hausdorff measures can become arbitrarily large. The point is, therefore, that to use these distances as measures of shape similarity, the sets X and Y must first be brought in the same region of space, e.g., by registration.

In the case of our experiments, however, this registration is not necessary, as we do compare the *proximity* of the snake to the ground truth, rather than shape similarity.

8.1.2 Algorithms

We measure the performance of the interpolation snakes model (Chapter 6), by measuring the proximity of the output snake to the target boundaries (ground truth) by using the measures described in the previous section. The significance of the results can be better understood relative to other models. As such, we compare head-to-head the following four algorithms:

CV: the **Chan-Vese** model, described in Section 5.2.2.

GAC: the **Geodesic Active Contours** model, described in Section 5.2.1.

KWT: the classical **Kass-Witkin-Terzopoulos** model, described in Section 5.1.

I-SNAKE: the **Interpolation Snakes** model, introduced in Chapter 6.

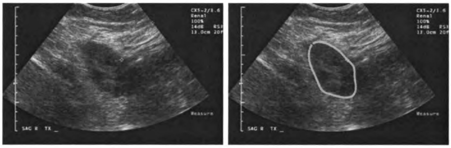
8.1.3 Data Sets

We test all four algorithms on two types of data: *ultrasound* images of human kidneys and *synthetic* and images with different amounts and different types of noise.

Ultrasound. Our kidney database consists of 117 images, each of size 733 × 471 pixels. Each kidney in this database was manually traced by a human expert[1]. These manually traced boundaries represent the *ground truth* (GT) for the ultrasound images. It has to be noted that, reliable as it may be, the kidney GT

[1]Dr. Dorry Segev, Division of Transplantation, Department of Surgery, Johns Hopkins Institutions.

is not 100% objective, because in several images the boundaries are so weak that the true boundaries are not distinguishable, even to the trained eye of a kidney surgeon.



(a) (b)

Figure 8.1: [a] Kidney ultrasound. [b] true boundaries (GT) manually traced by a human expert.

Synthetic images. We test our algorithms on synthetic images (in addition to ultrasound), in which the ground truth is known and is entirely *objective*. To this end, we create a family of binary images representing smooth, similar shapes as follows. We generate an i.i.d family of vectors $C^j = (c_1^j, \ldots, c_n^j) \in \mathbb{R}^{2n}$, $j = 1 \ldots K$ from a Gaussian distribution of mean C^0 and user-defined standard deviation σ. Each component c_i^j is a point in $\mathbb{R}^2$ (Figure 8.2 (a)). The mean C^0 is either manually defined by the user, or randomly generated. We interpolate each vector C^j by a *closed* spline S^j (Figure 8.2 (b)). The only condition we impose on S^j is that it does not self-intersect, so that the interior of the spline is well defined. A binary image I^j is then constructed for each contour S^j, in which the foreground is the interior of S^j, and represents the object of interest (Figures 8.2 (c)-(f)).

Each of the clear binary images I^j is contaminated with several types of noise, shown in Figure 8.3. We describe each of them below. In the sequel, the index j in I^j is irrelevant, so we drop it.

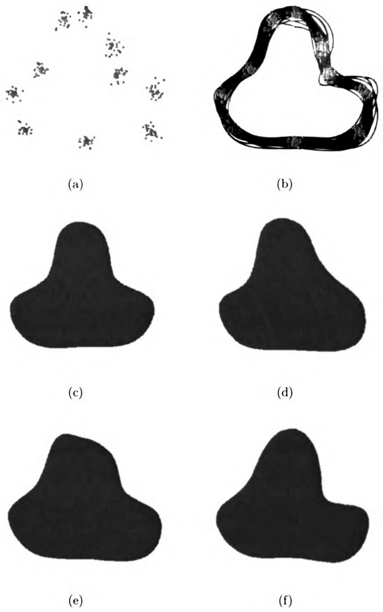


Figure 8.2: Synthetic images. (a) Clusters of control points. There are 30 points at each site, drawn from a Gaussian distribution of a user-defined standard deviation σ. (b) Interpolation splines using the control points from (a). (c)-(f) Binary images obtained as the interior of the first four splines in (b).

Gaussian noise. $I_{Gauss} = I + \eta_\sigma$ where at each pixel (x, y), the value $\eta(x, y)$ is drawn from a normal distribution of zero mean and standard deviation σ. See Figure 8.3 (b).

Uniform noise. We create the noisy image $I_{uniform}$ (Figure 8.3 (c)) by changing with probability p each pixel in the clear image I to a random gray level value. That is:

```
• for every pixel $z = (x, y)$ in $I$
    – draw a number $\rho \in [0, 1]$ from a uniform distribution.
    – define
```

$$I_{uniform}(z) = \begin{cases} random\ integer \in [0, 255] & if\ \rho < p \\ I(z) & otherwise \end{cases}$$

```
• end
```

Salt-and-pepper noise. This is nearly identical to the uniform noise, except that each pixel picked to be modified is replaced with either black or white (each of the two values with probability 50%), rather than with a random value in the full range $[0, 255]$. We denote the resulting image by I_{sp} (Figure 8.3 (d)).

Spread noise. This type of noise is obtained by swapping the value $I(z)$ at each pixel z with $I(z + dz)$ where dz is a displacement vector, drawn from a Gaussian distribution. of standard deviation σ. The pseudo code for this noising algorithm is the following:

```
• for each pixel $z$ in $I$ ($z \in \mathbb{R}^2$)
    – draw a vector $dz \in \mathbb{R}^2$ from a Gaussian distribution of zero mean,
      and standard deviation $\sigma$.
```

– define

$$I_{spread}(z) = \begin{cases} I(z+dz) & if\ z+dz\ inside\ I \\ I(z) & otherwise \end{cases}$$

This algorithm makes object boundaries in an image fuzzy, as some pixels close to the boundary, on one side of the boundary get swapped with pixels on the other side. See Figure 8.3 (e).

8.1.4 Snake Initialization

Generally, any active contour requires initialization. The outcome of the different models depends, sometimes to a great extent on the initial curve. Typically, an active contour application comes with a *graphical user interface* (GUI) in which one or more images are loaded and displayed, and in which the user initializes the snake manually, by marking a few points (ideally, as few as possible) near the boundaries of interest.

When running repeated, extensive tests on hundreds of images and trying semi-exhaustively different values for the parameters that control the snake evolution, manual initialization is prohibitive. We run our tests in a non-graphical application, and we provide different types of *automatic* initialization, having as starting point the ground truth itself. That is, given the ground truth (GT) in the form of an interpolation spline, we modify in certain controllable and repeatable ways the GT control points, then interpolate the modified points to produce the initial snake.

Needless to say, the ground truth is not used in any other way during the snake evolution; we only use it for initialization purposes, and only to avoid manual initialization on a large amount of data. We describe the initialization methods next.

Scaling initialization. This is perhaps the most straightforward way to initialize

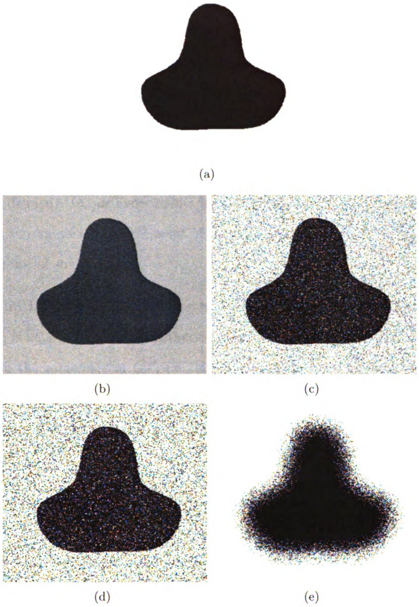


Figure 8.3: Various types of noise. (a) Clear image. (b) Gaussian noise with standard deviation $\sigma = 30$. (c) Uniform noise with probability $p = 0.3$. (d) Salt-and-pepper noise with probability $p = 0.3$. (3) Spread noise with standard deviation $\sigma = 30$.

the snake. We simply rescale the ground truth by a factor ρ, with respect to its own center of mass. Mathematically, if GT is the ground truth, z_0 is its center of mass, then the initial snake γ_0 is defined as

$$\gamma_0 = \rho(GT - z_0) + z_0$$

Gaussian randomization. In this case we "wiggle" the control points of the GT near their original location. That is, if c is any of the GT control points, we draw a point c' from a Gaussian distribution of mean c of standard deviation σ (user defined). In our experiments we test $\sigma = 10$ (near) and $\sigma = 20$ (far). We do this around each control point of the GT. The resulting interpolation spline is the initial snake.

Distance initialization. Here we construct the initial snake γ_0 such that any point on γ_0 is at a user specified distance d from the ground truth. All points on γ_0 are on the same side of GT, i.e., either all inside, or all outside of GT. The easiest way to do this is to build the *signed* distance map on the ground truth, and trace the d-level set on it.

Model initialization. If we have a template or a model μ of the object of interest, then we can use that model to initialize the snake, by warping μ onto the image, using a reduced number of *anchor points*. Specifically, in our kidney experiments we pre-define four anchor points on the model μ (Section 4.2). In a real application, the user would have to anchor the model μ to the image using the pre-defined anchors. After the user has defined the points in the image homologous to the anchors on the model, the warping is given by a *thin plate spline* (Section 3.2.1). This manual initialization of the anchors in the image would prevent us from running many batch jobs on all the ultrasound images. We therefore define the four anchors for each image, and store them once and for all, and use them in every trial (batch mode).

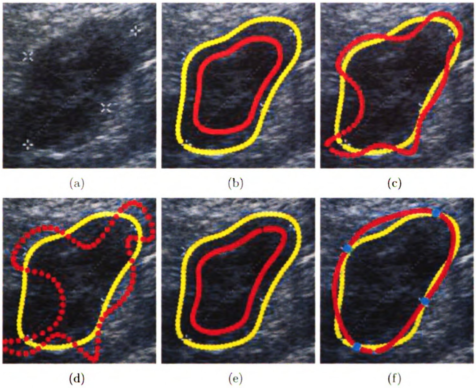


Figure 8.4: Different types of initialization. Yellow: ground truth (GT). Red: initial snake. (a) Clear image. The four points marked with a cross were defined by the sonographer when the image was acquired. (b) Scaling initialization by a factor of 0.7. (c) Gaussian randomization with a standard deviation $\sigma = 10$ pixels. (d) Gaussian randomization with standard deviation $\sigma = 20$ pixels. (e) Distance initialization at distance $d = 20$ pixels inside the ground truth. (f) Model initialization. The red curve is the image of a template kidney (not shown) via a TPS transform (Section 3.2.1) that maps four pre-defined anchors on the model (not shown) onto the four cyan anchors in the image (user-defined).

Each of these methods is significantly different from the others. The scaling and the distance initializations create a snake on one side of the ground truth (either all inside or all outside). This is a *sine qua non* condition in the case of geodesic active contours (GAC), due to the constant balloon force built into the model. This all-within or all-outside initialization is the only type that can be used with GAC. Observe, however, that the scaling and distance initialization methods differ in the fact that with scaling initialization, the distance (in pixels) between the initial snake and the ground truth will vary with the size of the object of interest. For instance, downsizing a small contour to 0.7 of its original size may produce an initial snake at distance of only 5 pixels, say, from the target boundaries, whereas downsizing a large contour by the same factor (0.7), may produce an initial snake at a significantly larger distance from the ground truth (20 – 30 pixels, or more).

Initialization by randomizing the control points of the ground truth is akin to how a human would initialize the snake manually, in a real application, and it would go on either side of the true boundary. A sloppy initialization would correspond to a larger standard deviation of the control points, whereas a more careful initialization would correspond to a small standard deviation.

Finally, the four-point, model-initialization might be the most convenient in a real application. We have observed that in some of the kidney images, four points were marked by the sonographer during the image acquisition: these were the antipodal points of the kidney along the long and the short axis. See Figure 8.4 (a). Since this operation seems to be common practice, warping the model onto the image using those four points as anchors comes at no extra cost for the medical personnel.

8.1.5 Parameters

Each of the four algorithms has a number of parameters that are crucial for the performance of the border detector. Setting the parameters is not a trivial task, because, to our knowledge, no active contours algorithm has an automatic way of adjusting the parameters. It is not possible (without an unreasonable amount of work) to determine all parameters for all algorithms by running exhaustive tests on all images in the database. Therefore, for each of KWT, CV and GAC, we determine empirically a reasonable set of parameters by using a small number of images, and use those parameters on the entire database.

However, for I-SNAKE, we determine the best set of parameters exhaustively, during testing. That is, we vary the most significant parameters within a specific range, in small increments, and test I-SNAKE on the entire database, *once for each set of parameters*. There are a number of parameters of the interpolation snakes, which we shall describe next:

Number of control points. This is one of the most important parameter, and controls the smoothness of the spline. The more control points, the more flexible the spline, and higher the time complexity. This parameter is determined semi-exhaustively, by testing $n = 10, 12, 14, 16, 18, 20$ control points.

Number of points on the spline. This parameter does not influence too much the performance, as long as the spline is dense after discretization. Theoretically, the number of points on the spline influences the image energy term (E_{img}). To remove this dependency, we normalize E_{img}, dividing it by the number of points on the spline.

Weights at the control points. These weights determine whether the snake should choose a light-dark edge, or viceversa. Since by and large, the interior of the kidney near the boundary is dark, and the exterior is light, it follows that for a snake oriented clockwise, the weights must have positive sign, so we define them as $+1$. Very rarely, the polarity of the kidney boundary happens to be reversed, due to image artifacts, in which case we initialize one or two weights to -1 (while the majority of the weights remain still $+1$).

Weights of the three energy terms: w_{img}, w_{arc} and w_{shape}, corresponding, respectively to E_{img}, E_{arc} and E_{shape} (see Equation (6.1)). These parameters control how much each of the three energy terms is to be trusted. We take them each within the range $[0, 100]$. The particular range is not important. What *is* important is that we use the same range for all three weights. Having the same range for all energy weights makes sense only if E_{img}, E_{arc} and E_{shape} have been normalized, say, to $[0, 1]$. We explain this normalization in detail in Section 8.2.4. It turns out that of the three weights, the most influential is the amount of shape prior. We have observed that increasing it beyond 40, the snake tends to ignore the image data, trying hard to accommodate the shape model. We run our tests with $w_{shape} = 0, 10, 20, 30$, while keeping $w_{img} = w_{arc} = 50$.

8.2 Ultrasound Results

We conducted extensive, systematic tests on the 117 images in our kidney database with all four algorithms, by varying the initialization method and various parameters. It is therefore not feasible to include all the detailed results in this presentation. Therefore, although each model was tested on all 117 images, we limit the presentation to 20 images per model, obtained with the parameters with the best performance. To further limit the size of the presentation, we cropped these images around the region of interest, and scaled them down to fit on a single page. In addition, to the 20 random images, for each of the algorithms we present the best, medium and worst cases, and possibly some special cases, roughly at the original resolution.

Table 8.1 summarizes the best results of all four algorithms.

Algorithm	Initialization/Parameters	NSD (%)	Hausdorff	Average Hausdorff
CV	GT scaling (0.7)	48.90%	82.6	39.394
GAC	GT scaling (0.7)	25.32%	49.18	16.14
KWT	GT rand, (std. dev. 10)	10.44%	17.10	5.93
I-SNAKE	**Model (4 anchors)**	**7.60%**	**14.32**	**4.90**

Table 8.1: Overall performance of the four active contour models tested on ultrasound images.

8.2.1 The Chan-Vese model

Recall (Section 5.2.2) that the Chan-Vese model tries to find - in a piecewise-constant, bi-modal grayscale image - two gray levels c_1 and c_2 and a curve γ (with possibly more than one connected component) that best approximate the image. That is, if u_0 is the given image, one tries to minimize the energy

$$E(c_1, c_2, \gamma) = \alpha \int_{int(\gamma)} |u_0(x) - c_1|^2 dx + \beta \int_{ext(\gamma)} |u_0(x) - c_2|^2 dx + \nu \cdot Length(\gamma) \tag{8.2}$$

A level set implementation of the minimization is possible if we embed, as usual, the curve γ as the zero level set of a surface ϕ, changing the energy (8.2.1) accordingly:

$$E(c_1, c_2, \phi | u_0) = \alpha \int_\Omega |u_0 - c_1|^2 H(\phi) + \beta \int_\Omega |u_0 - c_2|^2 (1 - H(\phi)) + \nu \int_\Omega |\nabla H(\phi)| \tag{8.3}$$

where Ω is the rectangle on which the image is defined, and H is the Heaviside function.

In our implementation, the evolving surface ϕ is updated over the entire image, so every iteration is an $O(n^2)$ process. Given the relatively large size of our ultrasound images (733 × 471), in order to keep the computation time reasonable, we downsize the image to 30%, and define a rectangular region of interest around the ground truth and restrict the evolution of ϕ over that region only. See Figure 8.5.

The algorithm was initialized by scaling down the ground truth by a factor of 0.7. However it should be mentioned that since the Chan-Vese model performs segmentation without a built-in edge detector (but rather by seeking the largest homogeneous regions), the snake is fairly insensitive to initialization. In a different experiment we initialized the snake as a circle centered at the center of mass of the kidney, with size comparable to the kidney size. The results were fairly similar.

Figure 8.5 shows the best result (top, $NSD = 14.9\%$), the worst (bottom, $NSD = 88.6\%$) and a result close to the average (middle, $NSD = 38.4\%$). The best result is obtained in an image without major gaps in the kidney boundary, and with a relatively homogeneous interior of the kidney. The middle images show one deficiency common to all active contours without shape prior knowledge: leakage through gaps in the boundary. The segmentation detects boundaries reasonably, but those boundaries happen not to be entirely the boundaries of the kidney. However, one cannot expect the snake to stop where there is no abrupt change in the intensity level. Refer to Figure 8.5 top and middle rows. At the places where the snake leaks, the regions inside and outside the ground truth are not distinguishable by their intensity (which the model computes), so for all the snake knows, those oversegmented regions are part of the target object.

The worst case occurs when the algorithm mistakenly detects the boundary of the white tissue inside the kidney, instead of the true kidney boundaries. The reason for this is that the algorithm is agnostic to the color of the object and the background. As long as there is a significant separation of two regions, the algorithm will detect it, whether the interior is light and the exterior is dark, or viceversa.

Figure 8.6 presents twenty randomly selected images with the border detected by the Chan-Vese model shown in red. These images are sorted in increasing order of the normalized symmetric difference (NSD). Figure 8.7 shows the same twenty images as in Figure 8.6 with the ground truth traced in yellow, for comparison. It appears that in about one third of the selected images the snake has detected the internal white tissue as the object of interest, so this situation is not uncommon.

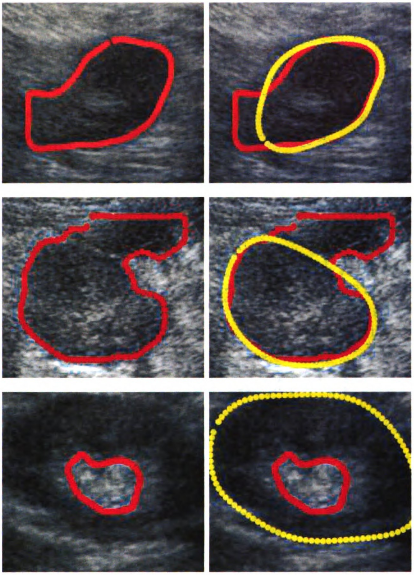


Figure 8.5: Segmentation of kidney ultrasound using the Chan-Vese model. Left column: binary segmentation of the region of interest defined around the ground truth. Right column: ground truth (yellow) and traced boundary between the segmented regions on the left (red). Top row: smallest normalized symmetric difference (NSD) error (14.9%). Middle row: $NSD = 38.4\%$. Bottom row: $NSD = 88.6\%$. Average $NSD = 40.8\%$.

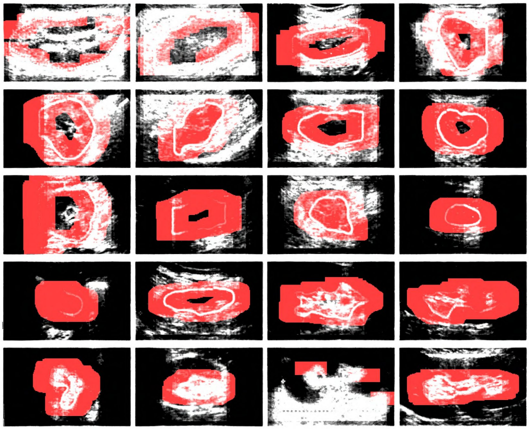


Figure 8.6: Chan-Vese model: twenty random samples. The samples are sorted in increasing order of the normalized symmetric difference (NSD).

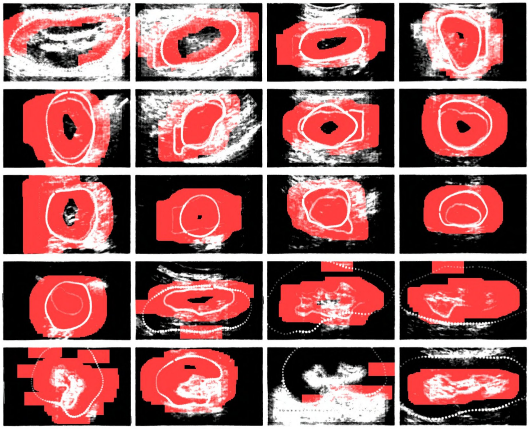


Figure 8.7: Chan-Vese model: the same images as in Figure 8.6 with the ground truth overlaid onto the image (yellow).

8.2.2 The Geodesic Active Contour Model (GAC)

The GAC model (Section 5.2.1) performs border detection by propagating a front (the snake) along its own normal with varying speed, defined so that the front propagation stops (or is extremely slow) at object boundaries. There is an implicit "balloon force" built into the model, which makes the front advance through homogeneous regions by "inflating" the snake. Near the edges, the balloon force should be superseeded by the "advection" force - another built-in component - which attracts the contour to edges in the image. If there are gaps in the boundary object, the balloon force remains dominant, however, so it keeps pushing the snake through the gaps, and the snake appears to leak through the boundary.

Any motion along the normal (Equation (5.6)) can be implemented using level set methods (Section 5.2). The most straightforward method is to solve the resulting evolution equation iteratively, over the entire image domain at each iteration (the full grid method). It turns out that it is sufficient to compute the updates only in a narrow band along the moving snake, which makes for fast level set methods. However, in our experiments we use the full grid method, as any of the fast methods is non-trivial to implement. Given the large size of our ultrasound images (733x471), we scale down the images to 30% and define a rectangular region of interest just around the kidney, in order to speed up the front propagation process. The results are brought back up to their original scale.

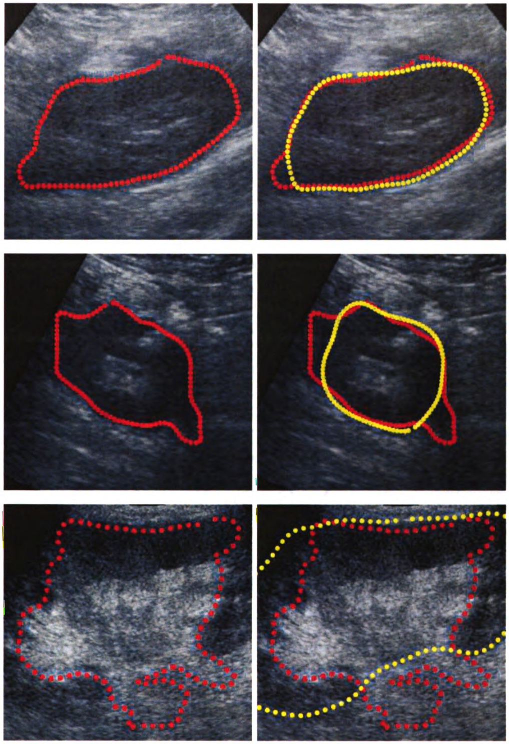


Figure 8.8: Segmentation of kidney ultrasound using the GAC model. Left column: output snake. Right column: ground truth (yellow) and output snake (red). Top row: smallest normalized symmetric difference (NSD) error (10.7%). Middle row: $NSD = 22.98\%$. Bottom row: $NSD = 49.28\%$. Average $NSD = 25.32\%$.

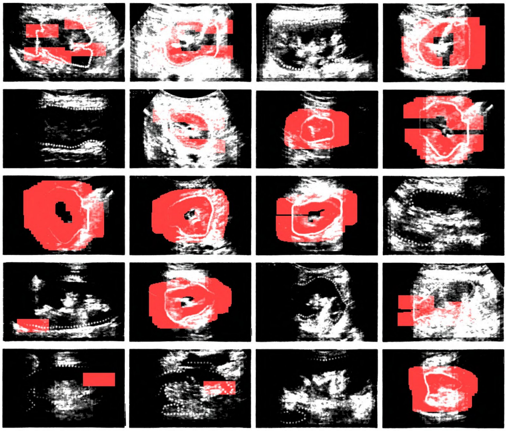


Figure 8.9: GAC model: twenty random samples. The samples are sorted in increasing order of the normalized symmetric difference (NSD).

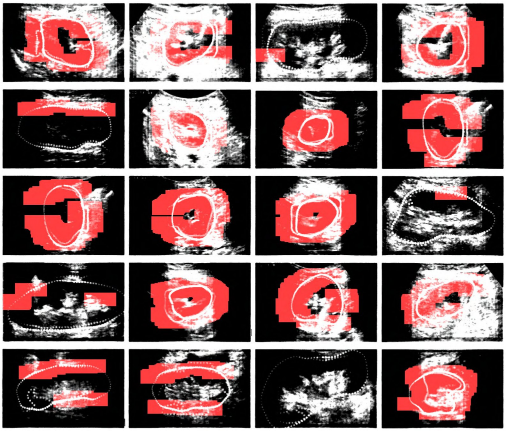


Figure 8.10: GAC model: the same images as in Figure 8.9 with the ground truth overlaid onto the image (yellow).

8.2.3 The Kass-Witkin-Terzopoulos Model (KWT)

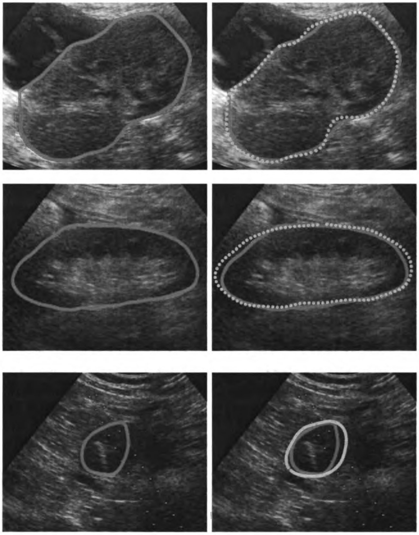


Figure 8.11: Segmentation of kidney ultrasound using the KWT model. Left column: output snake.Right column: ground truth (yellow) and output snake (red). Top row: smallest normalized symmetric difference (NSD) error (4.36%). Middle row: $NSD = 9.46\%$. Bottom row: $NSD = 27.22\%$. Average $NSD = 10.44\%$.

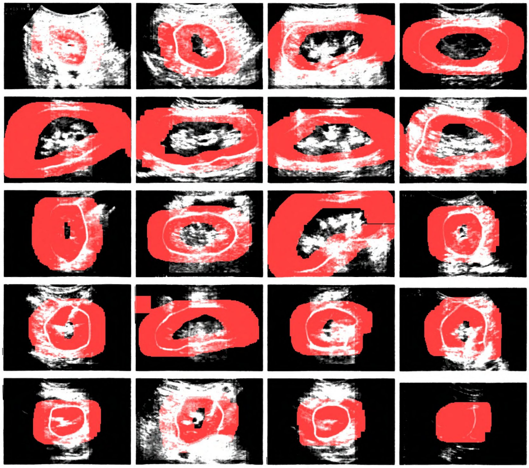


Figure 8.12: KWT model: twenty random samples. The samples are sorted in increasing order of the normalized symmetric difference (NSD).

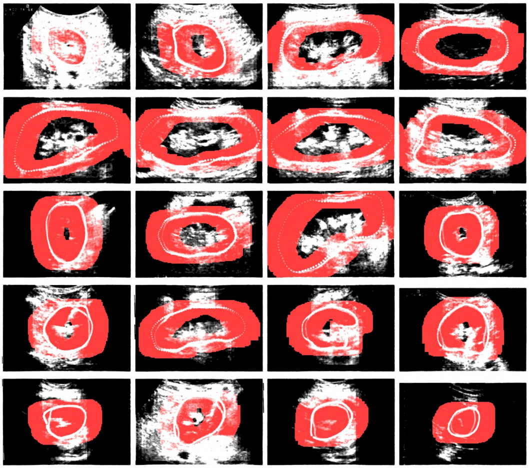


Figure 8.13: KWT model: the same images as in Figure 8.12 with the ground truth overlaid onto the image (yellow).

8.2.4 The Interpolation Snakes Model (I-SNAKE)

This section presents experiments based on the proposed interpolation snakes. We test extensively the effect of different parameters of the algorithm and different initialization methods. An exception is made, in that the results on *all* 117 ultrasound images are shown.

Recall (Sections 6.1 and 2.3) that with this method, the active contour is given by a *cubic interpolation spline* $\gamma = \gamma_C$ where $C = (c_1, \ldots, c_n)$ is the vector of control points that uniquely determine the spline γ. In all our tests γ will be a single-component closed curve.

Recall also from Section 6.1 (Equation (6.1)) that we fit the spline to the image, by minimizing the following energy functional:

$$\begin{aligned} E(\gamma) = -w_{img} \cdot & \underbrace{\int_\gamma w(s) \cdot \langle \nabla I^\perp, \frac{\gamma'}{|\gamma'|} \rangle \, ds}_{E_{img}} \\ & + w_{arc} \cdot \underbrace{\int_0^L | \, |\gamma'(t)| - 1 \, | dt}_{E_{arc}} \\ & + w_{shape} \cdot E_{shape} \end{aligned} \tag{8.4}$$

On the shape energy

In Section 6.2.3 we proposed a shape energy term of the form

$$E_{shape}(\gamma) = \int_\gamma s(\bar{C}(\tilde{\gamma}(s)))ds \tag{8.5}$$

where

- $s(x) = a^2x^2/(x^2 + b^2)$, for some constants a, b is a squashing function.

- $\bar{C}$ is the average chamfer transform of the *registered* training samples $\gamma_1, \dots, \gamma_k$, with base curve $\mu = \arg\min_\delta \bar{C}(\delta)$ (as in Section 4.2)

- $\tilde{\gamma}$ is the snake γ registered to the base curve μ by a *linear* transform $T_{\gamma,\mu}$. The aligning transform $T_{\gamma,\mu}$ must be computed *at each iteration* during the minimization process. It can be done fast (but sub-optimally), using semiaxes alignment (Section 3.4). Procrustes-type registration cannot be done, as we do not have point correspondences between an evolving snake and the template μ.

Remark 8.2.1 *We must mention at this point that we have done* extensive *tests using this way of computing the shape energy (i.e., by aligning the snake γ to the base curve μ by semiaxes registration). The results were reasonable, but contrary to our expectations, more or less on par with the traditional Kass-Witkin-Terzopoulos active contours (initialized sufficiently close to the target boundary, but with no prior knowledge). Both algorithms produced some* $9-10\%$ *normalized symmetric difference error (NSD).*

We looked long and hard for a reason, and we noticed that in some cases, the shape prior did more damage than good. We began to question the validity of our shape model, and after some thinking and experimenting, we realized that we had to allow for more general transforms when registering the snake γ to the model. The fact is that some kidneys differ from the model by a non-linear transform. In such cases, comparing the contour against the model by applying a similarity transform would incur an artificially large shape penalty on the snake. With a dominant shape term in the energy (8.4), *the image term would have little effect, resulting in a poor fit to the image.*

A remedy to this problem is to warp the shape model $\bar{C}$ (which is a surface in 3D) to the target boundaries, *before* evolving the snake. To this end, we pre-define four points (*anchors*) on the base curve μ. These only depend on the shape model, and therefore are the same across all test images. The user must supply a corresponding set of anchors in any test image, homologous to the model anchors. We compute next a *thin plate spline* T (Section 3.2.1) from the image space to the model space, which maps one set of anchors onto the other, and pull back the model $\bar{C}$ to the image, via T. The pull-back (denoted $T^*\bar{C}$) is defined pointwise, as

$$T^*\bar{C}(x,y) = \bar{C}(T(x,y))$$

Now we can compute the shape energy of the snake as

$$E^*_{shape} = \int_\gamma s(T^*\bar{C}(\gamma(s)))ds \tag{8.6}$$

where, as above, $s(x) = a^2x^2/(x^2+b^2)$ is the squash function.

Loosely speaking, the first shape energy (Equation (8.5)) means using an absolutee shape prior, i.e., *independent* of the data. By contrast, warping the shape model to the image amounts to some sort of *conditional* shape prior, given the data. Prior to these experiments, our *firm* expectation was that one should only use the absolutee form of shape prior. The results we present here (Tables 8.2, 8.3 and 8.4) are conclusive evidence to the contrary: if the shape model does not account for non-linear transformations present in the training objects, then the model should be made contingent upon the data (by warping). This fact may make sense after testing, but it was by no means obvious a priori. We consider this one good lesson of these experiments.

Weights and normalization

Notice that in Equation (8.4) we introduce explicitly three weights w_{img}, w_{arc} and w_{shape}, one for each energy term, in addition to the pointwise weights $w(\cdot)$ part of E_{img}.

Recall from Section 6.2.1 that, given an orientation along the curve, the weights

$w(\cdot)$ control whether the snake prefers an image gradient pointing to its left, or to its right, i.e., $w(\cdot)$ determine the polarity of the selected edge. Since for the most part (but not always!) the object of interest in ultrasound images is darker than its surroundings, we take all $w \equiv 1$.

The choice of the three energy weights must be done more carefully, for the simple fact that the three (un-weighted) energies take values in different ranges, so they are a priori non-comparable. In order for the three energy terms in (8.4) to be comparable, one should ensure that they have at least the same order of magnitude, if not the same range.

In the continuum, the gradient has infinite magnitude at a discontinuity. In the discrete case, however, $|\nabla I|$ is bounded by the magnitude (call it $|\nabla I|_{max}$ of the gradient at a perfect edge. In the case of gray level images, this bound is 255. Therefore, for the image term we have the following upper bound:

$$\begin{aligned} E_{img} &= \int_\gamma \langle \nabla I^{\perp}, \frac{\gamma'}{|\gamma'|} \rangle ds \\ &\leq \int_\gamma |\nabla I| ds \\ &\leq |\nabla I|_{\max} \cdot N_\gamma \end{aligned}$$

where N_γ is the number of points on γ. In all out experiments, $N_\gamma = 100$ points. Dividing by this a priori upper bound, E_{img} can be rescaled to the interval $[0, 1]$.

The internal energy

$$E_{arc} = \int_0^L ||\gamma'(t)| - 1| dt$$

is supposed to maintain the arclength parametrization of the snake. As such, it stays close to 0 throughout the minimization, so it can be assumed to already have range in $[0, 1]$, after dividing by N_γ.

Finally, the shape energy E_{shape} is given by the average chamfer transform (Equation (8.6)), squashed by the function $s(x) = a^2x^2/(x^2 + b^2)$. With $a = 1$, the range of s becomes precisely $[0, 1)$. The parameter b determines the width of the trough in the squashed average chamfer transform (Figure 6.9).

With all energy terms normalized to $[0, 1]$, we can take the weights all in one range, say $[0, 100]$. We have determined by trial and error (on a small number of test images) that in the absence of the shape energy $w_{shape} = 0$, the choice of $w_{img} = w_{arc}$ produces promising results. As long as w_{shape} remains zero, it does not matter what the common value for w_{img} and w_{arc} is. This value does matter though, when we have a non zero shape term. In our experiments we set $w_{img} = w_{arc} = 50$. We determine the value of w_{shape} semi-exhaustively, by trying $w_{shape} = 0, 10, 20, 30$. Larger values produce a snake too similar to the model, hence a poor fit to the image.

The outcomes of the interpolation snakes experiments are summarized in Tables 8.2 , 8.3 and 8.4, one for each of the three performance measures. Included in these tables are only the values for the interpolation snakes (I-SNAKE) and classical snakes (KWT), as the two level set algorithms (GAC and CV) were far behind due to border leakage (see Table 8.1).

- **Initialization** (columns 1-4):
  - Scaling. Controlling parameter: scaling factor.
  - Gaussian randomization of the control points. Controlling parameter: standard deviation.
  - Distance initialization. Controlling parameter: distance from the ground truth. The sign of the distance parameter indicates either interior initialization (negative sign) or exterior initialization (positive sign).
  - thin-plate-spline model initialization using four pairs of anchor points. The shape prior may or may not be used further during the minimization,

according to whether $w_{shape} > 0$ or $w_{shape} = 0$, respectively.

- $\mathbf{w_{shape}}$ (column 5): weight of the shape prior term E_{shape}.
- **I-SNAKE** (columns 6-7):
  - shape model a priori warped onto the image using a thin plate spline and four anchor points (shape prior contingent upon the image data).
  - un-warped shape model (shape prior independent on the image data).
- **KWT** (column 8).

Normalized Symmetric Difference (%)

Scaling	Stddev	Distance	Model	w_{shape}	Warped Shape Model	Absolute Shape Model	KWT
0.8	-	-	-	0	14.80	14.80	46.10
0.8	-	-	-	10	10.37	15.05	
0.8	-	-	-	20	9.63	15.72	
0.8	-	-	-	30	9.19	16.62	
-	10	-	-	0	9.38	9.38	10.44
-	10	-	-	10	7.95	9.42	
-	10	-	-	20	7.80	10.65	
-	10	-	-	30	8.29	12.71	
-	20	-	-	0	14.21	14.21	22.01
-	20	-	-	10	10.64	13.42	
-	20	-	-	20	9.57	14.08	
-	20	-	-	30	9.37	14.78	
-	-	-20	-	0	13.22	13.22	49.35
-	-	-20	-	10	9.58	12.73	
-	-	-20	-	20	9.07	14.06	
-	-	-20	-	30	8.96	15.31	
-	-	-	4 anchors	0	9.52		10.18
-	-	-	4 anchors	10	7.60		
-	-	-	4 anchors	20	8.08		
-	-	-	4 anchors	30	8.31		

Table 8.2: Interpolation snakes vs. KWT: normalized symmetric difference (NSD). Smaller NSD means better performance.

Hausdorff Distance (pixels)

Scaling	Stddev	Distance	Model	w_{shape}	Warped Shape Model	Absolute Shape Model	KWT
0.8	-	-	-	0	31.53	31.53	48.35
0.8	-	-	-	10	22.20	31.44	
0.8	-	-	-	20	19.71	31.88	
0.8	-	-	-	30	18.46	33.82	
-	10	-	-	0	18.37	18.37	17.10
-	10	-	-	10	15.69	18.72	
-	10	-	-	20	14.67	21.12	
-	10	-	-	30	15.68	26.10	
-	20	-	-	0	32.02	32.02	34.82
-	20	-	-	10	23.58	28.71	
-	20	-	-	20	20.31	29.08	
-	20	-	-	30	19.12	30.94	
-	-	-20	-	0	26.86	26.86	43.28
-	-	-20	-	10	20.05	25.32	
-	-	-20	-	20	18.05	28.18	
-	-	-20	-	30	17.10	29.98	
-	-	-	4 anchors	0	19.26		19.25
-	-	-	4 anchors	10	15.49		
-	-	-	4 anchors	20	15.26		
-	-	-	4 anchors	30	15.52		

Table 8.3: Interpolation snakes vs. KWT: Hausdorff distance (H). Smaller H means better performance.

Average Hausdorff Distance (pixels)

Scaling	Stddev	Distance	Model	w_{shape}	Warped Shape Model	Absolute Shape Model	KWT
0.8	-	-	-	0	9.56	9.56	28.47
0.8	-	-	-	10	6.87	9.96	
0.8	-	-	-	20	6.36	10.49	
0.8	-	-	-	30	5.96	11.33	
-	10	-	-	0	5.85	5.85	5.93
-	10	-	-	10	5.11	5.91	
-	10	-	-	20	4.98	6.65	
-	10	-	-	30	5.27	8.10	
-	20	-	-	0	8.74	8.74	12.06
-	20	-	-	10	6.65	8.37	
-	20	-	-	20	6.07	8.79	
-	20	-	-	30	5.92	9.47	
-	-	-20	-	0	8.03	8.03	28.83
-	-	-20	-	10	6.17	7.90	
-	-	-20	-	20	5.86	8.82	
-	-	-20	-	30	5.73	9.81	
-	-	-	4 anchors	0	5.97		6.21
-	-	-	4 anchors	10	5.08		
-	-	-	4 anchors	20	5.16		
-	-	-	4 anchors	30	5.28		

Table 8.4: Interpolation snakes vs. KWT: Average Hausdorff distance (AH). Smaller AH means better performance.

Perhaps the first thing to notice about the results reported in Tables 8.2, 8.3 and 8.4 is the consistency across the different performance measures, in the sense that for any of the three models tested, the ordering of any two experiments remains the same, regardless of the performance metric. It suffices, therefore, to comment (mostly) on the NSD table, say.

The only two cases when the classical model (KWT) competes with either interpolation snake counterparts is when the snake is initialized fairly close to the actual boundaries (normal distribution, with standard deviation of 10 pixels), or when initialized from the shape model. Even so, we find the performance of the KWT model quite remarkable. Based on Table 8.4, on average, in these two cases the KWT snake is within some 6 pixels from the ground truth, which is not bad for a snake with no shape memory.

The unexpected behavior of the interpolation snakes when the shape prior is not correlated with the image data can be seen in the "Absolute Shape Model" column: the performance[2] decreases with increasing values of the shape weight w_{shape}. In the best case (Gaussian initialization with standard deviation of 10 pixels and no shape prior), the interpolation snake with absolutee shape model has $NSD = 9.38\%$, only marginally better than KWT's 10.44%. Things change by a few points in favor of interpolation snakes when the "relative" shape prior is used, i.e., when the shape prior *is* anchored to the image.

Another noteworthy fact is that the interpolation snakes have a better performance than KWT's absolutee best (10.18%) by about 1 percent, even when initialized far from the ground truth: 9.19% and 8.96% NSD when initialized by scaling (0.8), and 20 pixels within, respectively. This means in a real application, the user would have more slack for initialization with interpolation snakes.

When the shape prior is used properly (i.e., contingent upon the image data), one

[2]Smaller numbers represent better performance.

observes the expected behavior (column 6): for any given initialization method, the performance achieves an optimum with some amount of shape prior, but increasing the shape weight beyond a certain point is detrimental.

Last, but not least, we have to remark that the parameters of the interpolation snakes can be better tuned for just about any of the images in the ultrasound database, producing better results. The problem is that the optimal parameters are not the same across *all* images. One image may require a certain amount of shape prior, while for another, no shape prior would produce the best results. Tables 8.2, 8.3 and 8.4 show the best (among non-optimal) parameters over the *entire* data set, not for each individual image. The lack of a uniform set of parameters can only be due, logically, to one of two causes: either *our* model is wrong (or at least incomplete), or there simply cannot be (for *any* model) a common set of parameters, optimal for all images. This dichotomy opens up a stream of philosophical questions, so for the time being we shall only confine ourselves with simply pointing out this problem, and not try to find an answer.

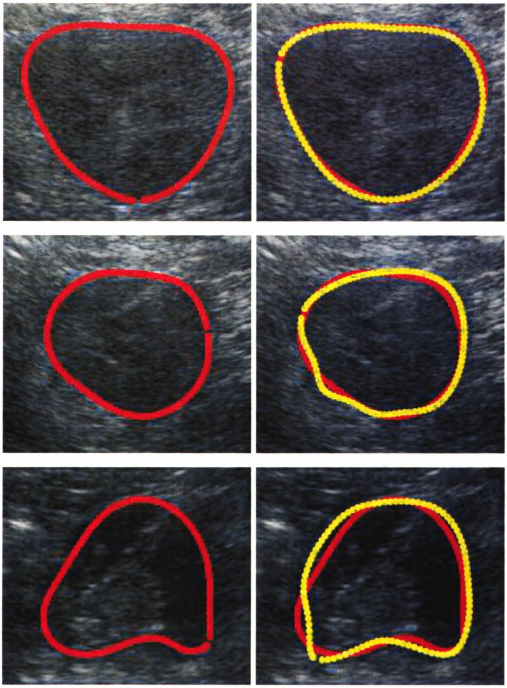


Figure 8.14: Segmentation of kidney ultrasound using the Interpolation snakes model. Left column: output snake. Right column: ground truth (yellow) and output snake (red). Top row: smallest normalized symmetric difference (NSD) error (3.71%). Middle row: $NSD = 7.72\%$. Bottom row: $NSD = 13.57\%$. Average $NSD = 7.60\%$.

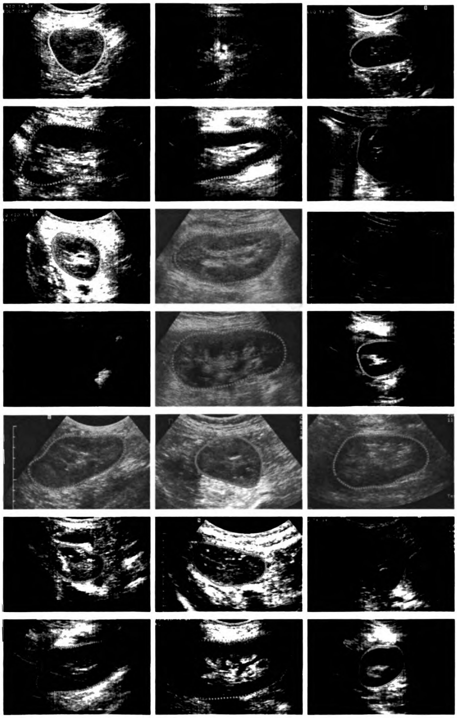


Figure 8.15: Detected contours. $3.71 \leq NSD \leq 5.52(\%)$.

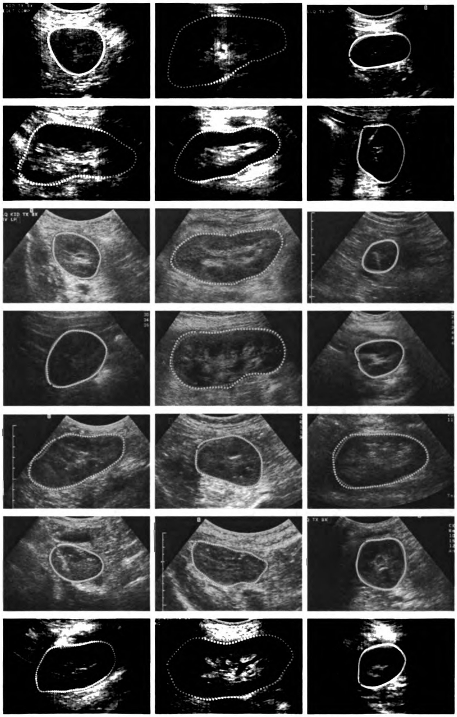


Figure 8.16: Red: detected contours (same as in Figure 8.15). Yellow: GT. $3.71 \leq NSD \leq 5.52(\%)$.

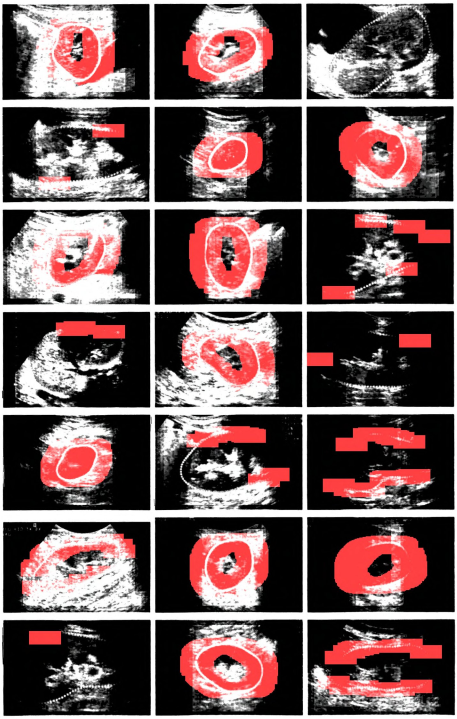


Figure 8.17: Detected contours. $5.52 < NSD \leq 6.65(\%)$.

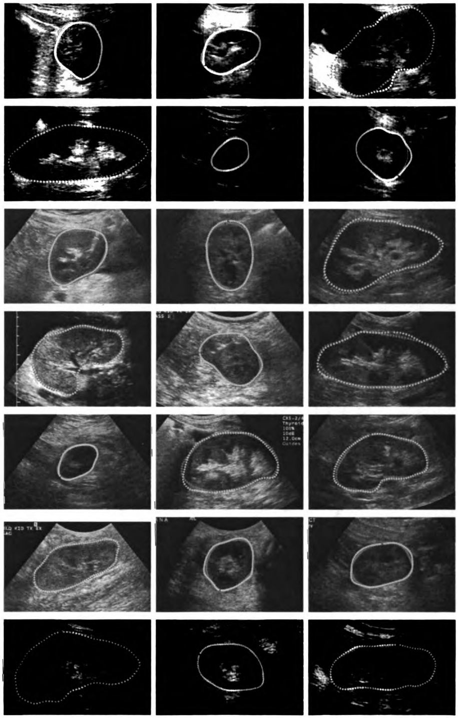


Figure 8.18: Red: detected contours (same as in Figure 8.17). Yellow: GT. $5.52 \leq NSD \leq 6.65(\%)$.

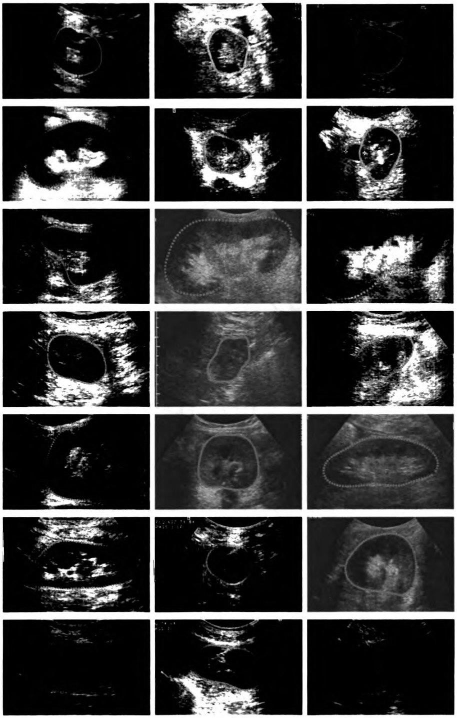


Figure 8.19: Detected contours. $6.65 \leq NSD < 7.87(\%)$.

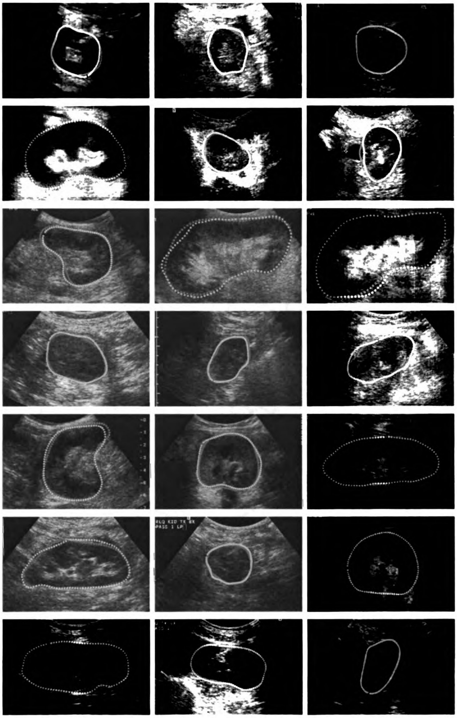


Figure 8.20: Red: detected contours (same as in Figure 8.19). Yellow: GT. $6.65 NSD \leq 7.87(\%)$.

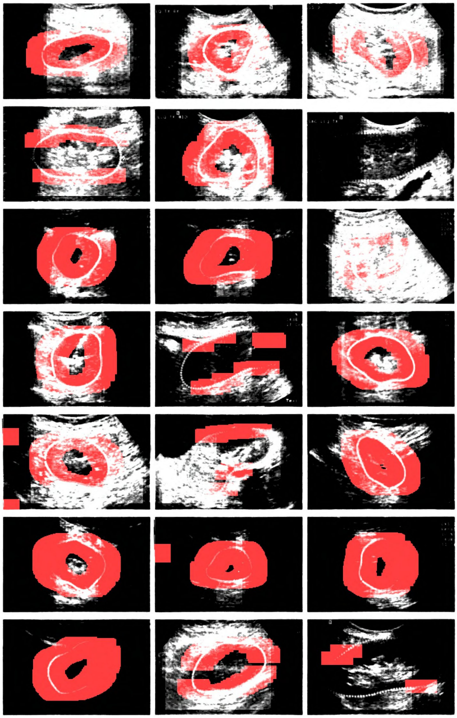


Figure 8.21: Detected contours. $7.87 < NSD \leq 8.79(\%)$.

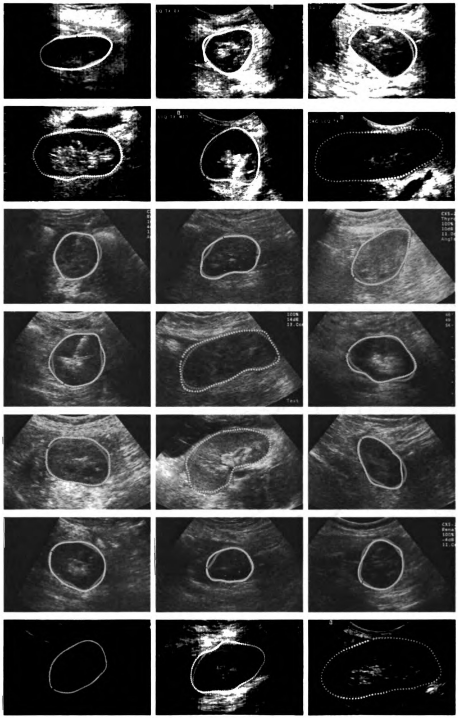


Figure 8.22: Red: detected contours (same as in Figure 8.21). Yellow: GT. $7.89 < NSD \leq 8.79(\%)$.

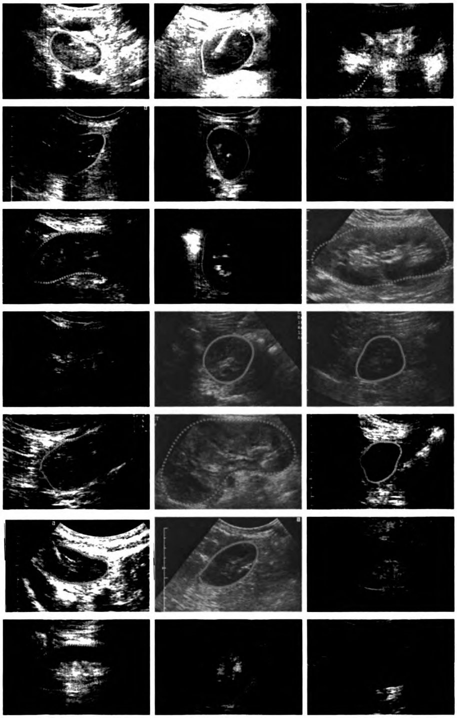


Figure 8.23: Detected contours. $8.79 \leq NSD < 9.91(\%)$.

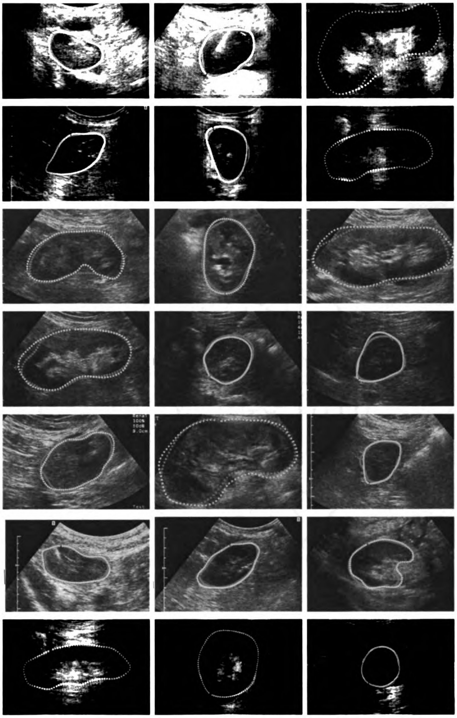


Figure 8.24: Red: detected contours (same as in Figure 8.23). Yellow: GT. $8.79 \leq NSD < 9.91(\%)$.

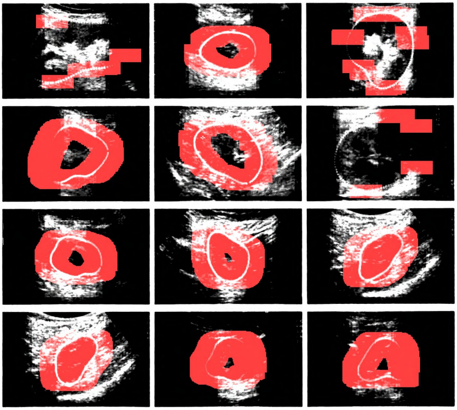


Figure 8.25: Detected contours. $9.91 \leq NSD \leq 13.57(\%)$.

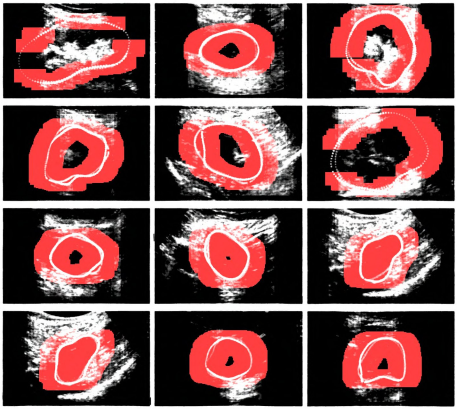


Figure 8.26: Red: detected contours (same as in Figure 8.25). Yellow: GT. $9.91 \leq NSD \leq 13.57(\%)$.

8.3 Blind study

To confer clinical validity to the interpolation snakes algorithm (Section 6.1) we devised a blind evaluation experiment, in addition to the objective comparison against the ground truth presented in the previous sections.

Specifically, in this experiment a human expert [3] compares the boundaries detected by several algorithms, without knowing which is which. For each of the 117 images used in our tests we draw three types of contours, each contour on a separate copy of the image, as follows:

- interpolation snakes (I-SNAKE) using the manual four-point model initialization ($NSD = 7.95\%$; see Table 8.2).
- the classical snakes (KWT) initialized randomly around the ground truth with a standard deviation of 10 pixels ($NSD = 10.44\%$).
- the ground truth itself (GT).

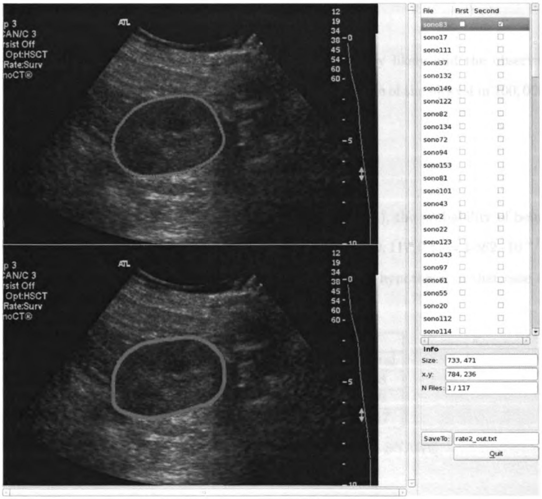
For each kidney ultrasound image I we have three *pairs* of images: $(I_{I-SNAKE}, I_{KWT})$, $(I_{I-SNAKE}, I_{KWT})$ and (I_{KWT}, I_{GT}). We generate the complete list of all 351 possible pairs, shuffle it and display each pair in turn in a graphical interface, shown in Figure 8.27. The interface never makes reference to the names of the algorithms whose results are being displayed.

We built the application trying to minimize the judge's time and effort. The user can toggle between the top and bottom images selecting this way the better of the two, either by clicking with the mouse, or using the up/down keys on the keyboard. The thickness of the contours is adjustable and can be controlled by the user with the left/right keys. A thickness of 0 is permitted, in order to allow the user to view the image without any contours drawn. Also, the contour marked as the better of

[3] Dr. Dorry Segev, Division of Transplantation, Department of Surgery, Johns Hopkins Institutions.

the two is drawn thicker than the other one, for easy visualization. Once a selection has been made, the user advances to the next pair. When ready, the selections are saved in a plain ASCII file with one row per pair, and with two columns. The winner of each pair is listed in the first column.

The task was clearly explained to the human expert. The ease of use of the application did not require any training session prior to the actual test. No time constraints were imposed.



File	First	Second
sono83	■	☑
sono17	☐	☐
sono111	☐	☐
sono37	☐	☐
sono132	☐	☐
sono149	☐	☐
sono122	☐	☐
sono82	☐	☐
sono134	☐	☐
sono72	☐	☐
sono94	☐	☐
sono153	☐	☐
sono81	☐	☐
sono101	☐	☐
sono43	☐	☐
sono2	☐	☐
sono22	☐	☐
sono123	☐	☐
sono143	☐	☐
sono97	☐	☐
sono61	☐	☐
sono55	☐	☐
sono20	☐	☐
sono112	☐	☐
sono114	☐	☐

Size: 733, 471
x,y: 784, 236
N Files: 1 / 117
SaveTo: rate2_out.txt

Figure 8.27: Graphical user interface for the blind study comparison of three types of kidney boundaries (I-SNAKE, KWT and GT). A human expert chose between outputs of three algorithms, displayed two at the time.

Tables 8.5 and 8.6 show how many times the I-SNAKE output was preferred over KWT and over GT, respectively.

Somewhat surprisingly, the interpolation snakes won over GT 80 times out of 117 (Table 8.6). Under a null hypothesis that both I-SNAKE and GT are equally likely to win ($p = 0.5$), the probability of *observing* (due to chance) 80 I-SNAKE outcomes out of 117 is given by a binomial distribution $b(k; n, p) = \binom{n}{k} \cdot p^{n-k} \cdot p^k$. In our case, with $n = 117, k = 80$ and $p = 0.5$ we get $b = 0.00004359$, or about 4 in $100,000$. We are, therefore, justified to reject the null hypothesis because there are only two logical possibilities:

- either it is true that I-SNAKE and GT are equally likely and the observed outcome of 80 I-SNAKE winners out of 117 total is one of the lucky 4 in $100,000$, *or*

- the null hypothesis is false to begin with.

Similarly, in the case of I-SNAKE vs. KWT (Table 8.5), the probability of being wrong in rejecting the null hypothesis is even smaller: $b(99; 117, 0.5) = 4.862 \cdot 10^{-15}$ (zero within machine precision), so the we reject the null hypothesis in that case as well.

winner	loser		
	KWT	I-SNAKE	Total
KWT	0	18	18
I-SNAKE	99	0	99
Total	99	18	117

Table 8.5: Blind comparison: KWT vs. I-SNAKE

The complete results of the blind study are shown in Table 8.7. For each type of contour (GT, KWT and I-SNAKE) there are three rows. To fix the ideas, let us look at the I-SNAKE algorithm as winner. The top row shows how many times I-SNAKE

winner	loser		
	GT	I-SNAKE	Total
GT	0	37	37
I-SNAKE	80	0	80
Total	80	37	117

Table 8.6: Blind comparison: GT vs. I-SNAKE

won over GT and KWT: 80 and 99 times, respectively, making for a total of 179 wins out of a 351 comparisons. The middle and the bottom rows represent the row percentage and the column percentage of the entries in the top row in each group of three. Thus, for instance, I-SNAKE won 179 times out of 351 comparisons (51.00%). Of these wins, 80 were over GT (44.69%) and 99 were over *KWT* (55.31%). Reading on columns now, I-SNAKE *lost* a total of only 55 comparisons out of 351 (15.67%) of which 37 times it lost to GT (67.27%) and 18 times to KWT (32.73%).

winner	loser			
	GT	KWT	I-SNAKE	Total
GT	0	82	37	119
row percentage	0.00	68.91	31.09	100.00
column percentage	0.00	45.30	67.27	33.90
KWT	35	0	18	53
row percentage	66.04	0.00	33.96	100.0
column percentage	30.43	0.00	32.73	15.10
I-SNAKE	80	99	0	179
row percentage	44.69	55.31	0.00	100.00
column percentage	69.57	54.70	0.00	51.00
Total	115	181	55	351
row percentage	32.76	51.57	15.67	100.00
column percentage	100.00	100.00	100.00	100.00

Table 8.7: Blind comparison: complete results. On rows, for each of GT, KWT and I-SNAKE categories: frequencies (top), row percentages (middle), column percentages (bottom).

8.4 Experiments on Synthetic Images

We leave now the ultrasound domain and present the tests on synthetic images. Table 8.8 summarizes the numerics of the four algorithms (CV, GAC, KWT and I-SNAKE) each under four different types of noise and under different initializations. It is similar to Tables 8.2, 8.3 and 8.4, and we shall not describe its elements in detail.

In general, the results are far better on synthetic images than on ultrasound, despite high levels of noise. In the case of CV, GAC and I-SNAKE, there is a significant decrease in the performance under spread noise, than compared with Gaussian, salt-and-pepper and uniform noise, which can be attributed to the fact that all the algorithms have a built-in smoothing filter, which removes to a large extent the gaussian, the salt-and-pepper and the uniform noise, but are not as effective on the spread noise.

When initialized at some distance from the target boundaries, KWT snakes perform very poorly, with or without noise, and this should come as no surprise. When initialized randomly, with standard deviation of 10 pixels from the true boundaries, the classical snakes (KWT) perform almost perfectly under all noise conditions, and we explain this based on the proximity to the ground truth more than anything else.

Note that initialization at about 10 pixels from the ground truth produced also the best KWT results in ultrasound images as well (but no better than I-SNAKE). This observation reiterates the fact that the apparently good performance of KWT in ultrasound is due not so much to the merits of the algorithm per se, but rather to initialization. The interpolation snakes have better performance than KWT when initialized within 10 pixels from the ground truth, or when using the 4-anchor model initialization method, but also have comparable performance to KWT even when initialized at a significant distance from the ground truth (either at distance 20 pixels from the ground truth, or by a 80% downscaling of the true boundaries). Compare Table 8.8 with Tables 8.2, 8.3 and 8.4).

Algorithm	Noise	Scaling	Rand	NSD (%)	Hausdorff	Av. Hausdorff
CV	gaussian	0.7		3.4	7.83	3.31
	s&p			3.58	8.46	3.54
	uniform			3.10	7.91	3.29
	spread			9.81	17.69	6.76
GAC	gaussian	0.7		5.97	10.47	4.44
	s&p			7.84	11.25	5.52
	uniform			7.06	10.10	5.08
	spread			16.22	22.67	10.60
KWT	gaussian	0.7		68.02	86.82	56.09
	s&p			68.11	86.39	56.18
	uniform			68.78	87.02	56.94
	spread			62.67	85.11	50.66
KWT	gaussian		10	1.62	8.09	2.71
	s&p			1.58	8.10	2.69
	uniform			1.55	8.01	2.69
	spread			1.41	8.03	2.65
I-SNAKE	gaussian	0.7		2.16	5.63	2.82
	s&p			2.64	7.60	3.00
	uniform			2.36	6.78	2.87
	spread			5.94	13.01	4.74

Table 8.8: Results of the active contour algorithms on synthetic images, under different conditions of noise (column 2), and initial conditions (columns 3 and 4).

We present next typical images for each of the 5 categories in Table 8.8. See Figures 8.28, 8.29, 8.30, 8.31 and 8.32. In each case, the images are arranged in a 4×3 display. On each row there is a different type of noise. The left column shows the initial snake, the middle column shows the final result, and the right column shows also the final result (red) with the ground truth overlaid on top (yellow).

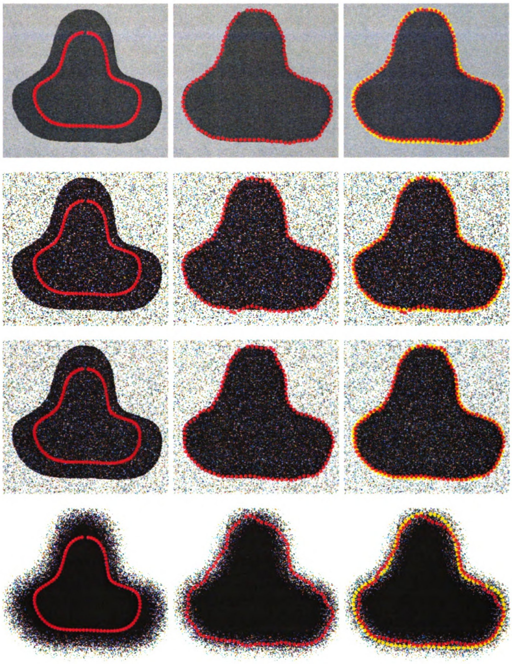


Figure 8.28: **Chan-Vese model on synthetic images.** Left column: initial snake (scaling to $0.7 \cdot GT$). Middle column: final snake. Right: final snake (red) and GT (yellow). On rows, top to bottom: Gaussian noise, salt-and-pepper noise, uniform noise, spread noise. Original images were cropped and scaled down for display purposes, so some noise was lost in the process.

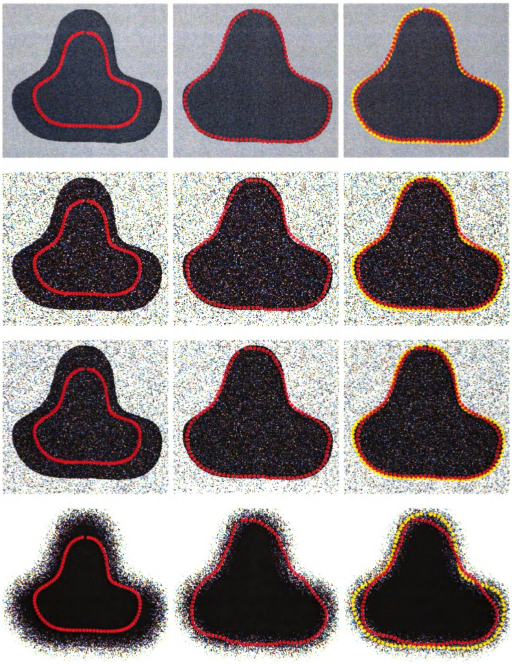


Figure 8.29: **Geodesic Active Contours model (GAC) on synthetic images.** Left column: initial snake (scaling to $0.7 \cdot GT$). Middle column: final snake. Right: final snake (red) and GT (yellow). On rows, top to bottom: Gaussian noise, salt-and-pepper noise, uniform noise, spread noise. Original images were cropped and scaled down for display purposes, so some noise was lost in the process.

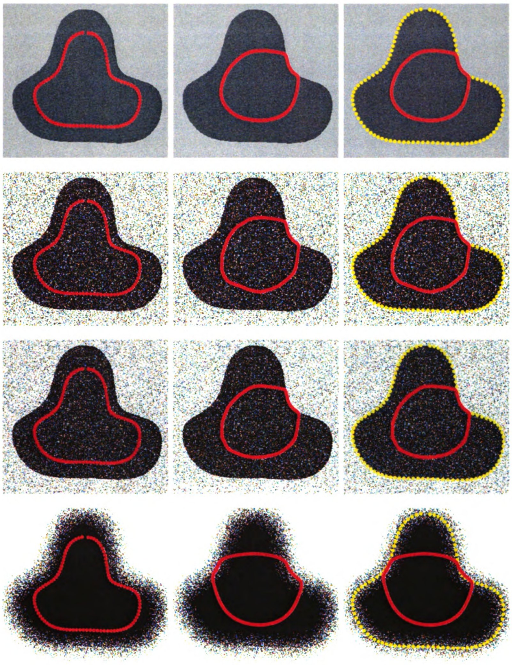


Figure 8.30: **Kass-Witkin-Terzopoulos model (KWT) on synthetic images.** Left column: initial snake (scaling to $0.7 \cdot GT$). Middle column: final snake. Right: final snake (red) and GT (yellow). On rows, top to bottom: Gaussian noise, salt-and-pepper noise, uniform noise, spread noise. Original images were cropped and scaled down for display purposes, so some noise was lost in the process.

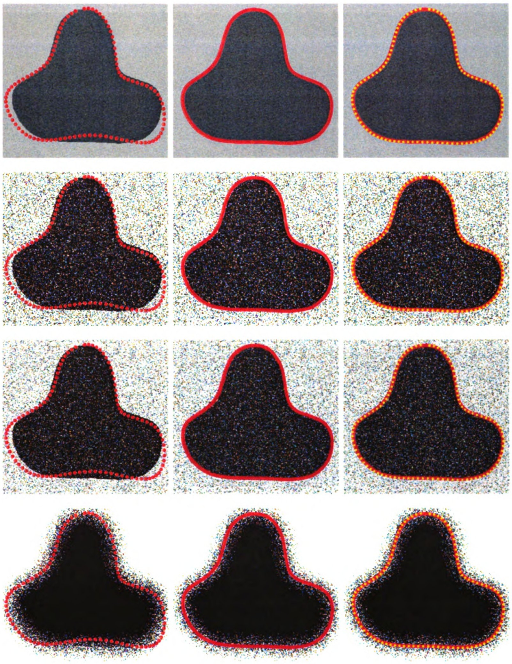


Figure 8.31: **Kass-Witkin-Terzopoulos model (KWT) on synthetic images.** Left column: initial snake (Gaussian randomization with standard deviation $\sigma = 10$ pixels). Middle column: final snake. Right: final snake (red) and *GT* (yellow). On rows, top to bottom: Gaussian noise, salt-and-pepper noise, uniform noise, spread noise. Original images were cropped and scaled down for display purposes, so some noise was lost in the process.

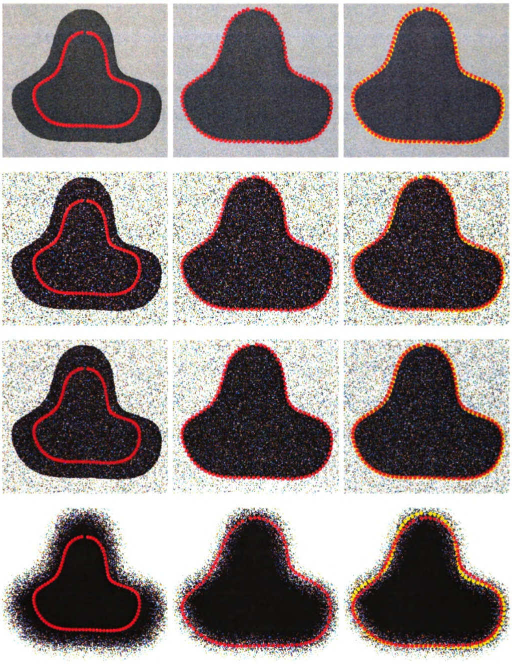


Figure 8.32: **Interpolation snakes model (I-SNAKE) on synthetic images.** Left column: initial snake (scaling to $0.7 \cdot GT$). Middle column: final snake. Right: final snake (red) and GT (yellow). On rows, top to bottom: Gaussian noise, salt-and-pepper noise, uniform noise, spread noise. Original images were cropped and scaled down for display purposes, so some noise was lost in the process.

8.5 Border Detection Using 1D Break-Point Detection.

We return in this section to border detection in ultrasound, using this time the 1D break-point detection algorithm from Chapter 7. The process involves the following steps:

1. Manually define a region of interest (ROI), by drawing a circle around the target object, centered roughly at the object center of mass. Move outward along equally spaced rays from the center of the ROI. In our application we used 5° between rays. The intensity profile along any of the rays is obtained by averaging over a sector of a certain angle (user defined).

2. Apply the 1D break point detection algorithm from Chapter 7 with a single break point, along each of the radial intensity profiles (Figure 8.33 (a)). We are only interested low-to-high intensity points (marked with red), as the kidney interior is in general darker than the background.

3. Some of the (correctly) detected break points are not on the target boundary. We try to eliminate them by using a shape model, in several steps. First, fit the shape model to the red points using non-isotropic similarity transforms (translation, rotation, and different scales in the x and the y direction, but no shear). The measure of fit is given by the chamfer transform on the red points. See Figure 8.33 (b).

4. Define a tubular neighborhood along the fitted model (Figure 8.33 (c)). The width of the tube is proportional to the standard deviation of all the red break points from step 1. Extract intensity profiles within the tube, along segments normal to the fitted model.

5. Re-compute the break points on the newly defined intensity profiles (Figure 8.33 (d)).

6. If need be, repeat steps 3 to 5.

7. Use a *smoothing spline* [29] to approximate the object boundary, based on the detected break points (Figure 8.33 (e)). We employed a smoothing spline algorithm that automatically determines the breaks and the control points. It only depends on a single parameter called *smoothing factor*. When the smoothing factor is zero, the smoothing spline goes through all the data points, being nothing more than an interpolation spline. As the smoothing factor increases, so does the smoothness of the spline, at the expense of missing the data points.

The results are promising in many cases. Sometimes the fitted model (step 3) is so good that the tubular neighborhood refinement and the smoothing spline are not necessary. When the tube *is* necessary though, it must be sufficiently wide, to cover completely the target boundary. This condition, in turn, depends on how close the model fitted to the break points (step 3) comes to the target boundary (not to the breakpoints!). If in some regions, the model is too far from the kidney boundary, the tube will miss that region, and no break point will be detected there at step 5. Last, but not least, we have observed experimentally that the smoothing factor of the final spline varies greatly from case to case. It may be possible to solve this latter problem by computing a family of splines over an entire discrete range of the smoothing factor, and by choosing the optimal one, with respect to some optimality criterion (yet to be determined).

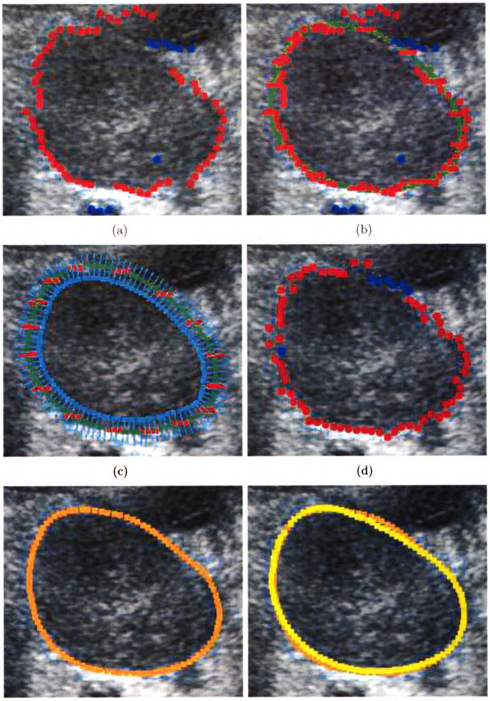

(a) (b) (c) (d) (e) (f)

Figure 8.33: 2D border detection using radial 1D break-point detection. (a) Break points along 72 rays starting from a user-defined center. The actual intensity profiles not drawn. Red: dark-to-light break point. Blue: light-to-dark break point. (b) Shape model (green) fit to the red break points. (c) Tube (cyan) across the fitted model (green). (d) Break points on the tube intensity profiles. (e) Smoothing spline on the red points from (d). (f) Smoothing spline (beige) and ground truth (yellow).

8.6 Miscellaneous Images

We present here an application of the interpolation snakes framework to real-life images. Since we are not using any full fledged general purpose object detector, manual initialization of the snake is still required.

Specifically, we test the interpolation snakes on eight images of a truck with varying background and occlusion. As with the kidney images, we first build a model. Since this is not a comprehensive test, we only build a fast and frugal model: we simply take as our model the distance map built on a manually traced truck contour in just one of the eight images. The reason this is justified is that the shape variations can be described mostly by affine transforms (no non-linear shape variations, as is the case with kidneys), as the truck is a rigid object, photographed in near profile. The affine transforms are factored out by the registration process that takes place at every iteration during the snake evolution.

For the results, refer to Figures 8.34 (initialization) and 8.35 (output). Observe, in the first place, that detection of a light object on a dark background using a clockwise oriented snake requires negative weights. This is being visualized in the images by marking the control points with a red minus sign. In the case of kidneys, the object was generally darker than the background, and the control points were all positive.

Each of the bottom four images contain occlusion, which the snake with shape memory can handle successfully. Also, the snake seems remarkably robust, remaining undisturbed by the trees in the background. This is one application which can lead to a spin-off of our work in ultrasound, aimed at the recognition and the tracking of vehicles.

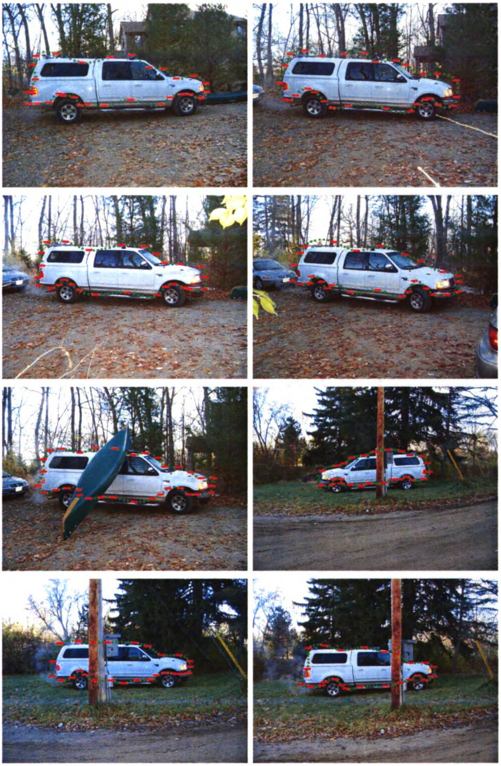


Figure 8.34: Interpolation snakes on truck images. Initialization of interpolation snakes.

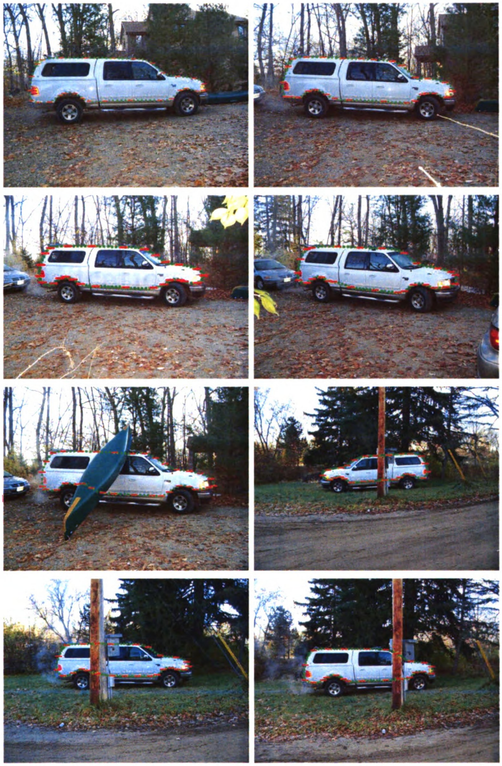


Figure 8.35: Interpolation snakes on truck images. Output of interpolation snakes.

Chapter 9

Model Fitting For Fingerprint Classification

We present in this chapter a simplified version of the work that we did in the area of fingerprint classification [78]. It was done prior to our work on active contours, so it does not involve interpolation snakes. However, chronologically and conceptually this work is the precursor of our snakes, as it is still based on the fitting of a shape model to an image. The nature of the problem requires that the shape remain roughly unchanged for classification purposes, so the fitting is done via affine transforms, rather than by free-form deformations as in the case of snakes.

9.1 Problem Definition

Fingerprints are biometric characteristics of humans, that consist of ridges and furrows at the tips of the fingers. There is evidence [37] that human awareness of these patterns predates Christianity. Since the last century fingerprints have been used systematically for authentication, particularly in criminal investigation. Based on empirical evidence [37, 38] it is widely accepted that fingerprints uniquely identify an individual. Unlike other authentication methods (such as face or voice recognition, which may not distinguish identical twins), fingerprint authentication is unequivocal. Moreover, fingerprints do not change over time, which gives them an important advantage over other biometric measurements. Lastly, in comparison to other biometrics techniques (such as retinal scan and iris scan), fingerprint acquisition is relatively simple and inexpensive. All these characteristics make the use of fingerprints one of the most pervasive methods of authentication, both in the forensic sector as well as in the commercial sector (e.g. access control).

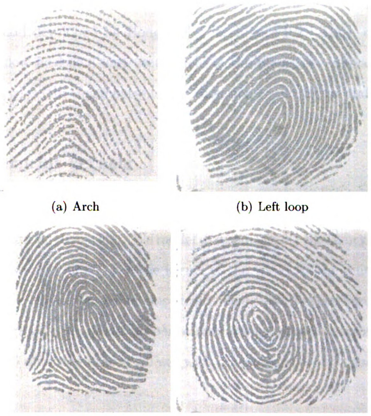
Given the astronomical size[1] of fingerprint databases, it became apparent that an indexing methodology which would assign fingerprints into a small number of categories was in order. The system developed at the end of the 19-th century by Sir Edward Henry is the basis of modern automatic fingerprint identification systems (AFIS) in the majority of English-speaking countries, and serves the purpose of what is usually termed as (primary) classification of fingerprints [13, 51, 62, 60]. The system adopted by the FBI contains eight basic fingerprint patterns [37, 38]: arch, tented arch, left loop, right loop, whorl, central pocket loop, double loop and accidental. We shall confine ourselves here to only four major classes of fingerprints [2]: arch (A), left loop (L), right loop (R) and whorl (W). See Figure 9.1.

There have been several methods proposed for fingerprint classification, including (but not limited to) syntactic methods [90], singularity based methods [62, 64], and neural network methods [12]. For a more detailed literature review see for instance [13, 51, 60] and the references therein.

We address the problem of automatic fingerprint classification into one of the 4 types (A, L, R, W) using a novel method which we term *kernel fitting*, and provide detailed classification results.

[1]There were 810,188 records (containing 10 fingerprints each) upon the formation of the Identification Division of the FBI in 1924 [37]. Currently there are 226 million cards on file, representing about 79 million individuals [36].

[2]However, the method we propose can be readily extended to more classes of fingerprints.


(a) Arch (b) Left loop

(c) Right loop (d) Whorl

Figure 9.1: Four major classes of fingerprints.

9.2 Mathematical model

9.2.1 Flow field

We model the ridges of a fingerprint as curves in the plane, and define the *flow field* as the direction of the tangent to these curves at each point [3]. Several methods have been proposed in the literature for the extraction of the flow field [17, 63, 64]. We employ the method developed by Hong [59]. See Figure 9.2. Thus, conceptually, we think of the flow field as a unit vector field in the plane, or, equivalently, as its argument $\beta \in [0, \pi)$ [4] (the angle between the direction of the vector and the x-axis).

9.2.2 Fingerprint kernels

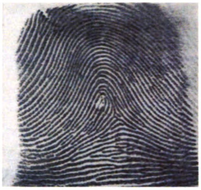
We *define* class-specific *kernel curves* based on the original definition of fingerprint classes of Sir Edward R. Henry [49]. We model mathematically Sir Henry's definitions of fingerprint classes as follows. Let the kernel of the whorl be the unit circle. The kernels of the other classes we define empirically, using splines (see Figure 9.3). For each class we learn a *kernel model* by performing a generalized Procrustes analysis (Section 4.1) on 20 manually traced splines with homologous control points.

9.2.3 Kernel fitting

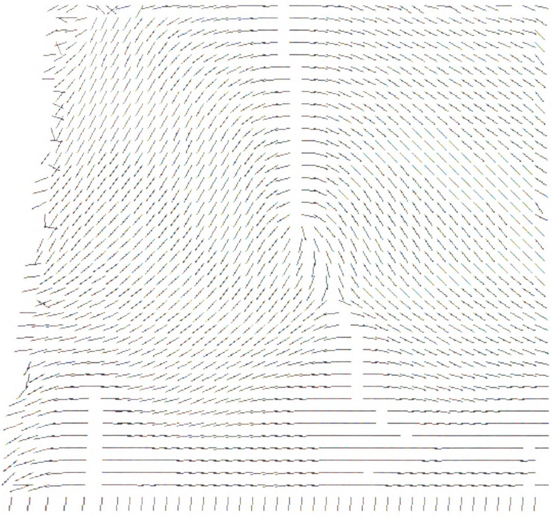
Suppose we have a smooth vector field V defined over some region in the plane $\mathbb{R}^2$. Let $\gamma(t) = (x(t), y(t))$ be a parametric curve in $\mathbb{R}^2$, let $\dot{\gamma}$ be the tangent to γ, and α its argument. As before, let β be the argument of V. See Figure 9.5.

Define an energy functional that captures the difference between the direction of $\dot{\gamma}$ and that of the vector field V at the point $\gamma(t)$ by

[3] Except at a small number of points, where the curve is not smooth (e.g. bifurcations).
[4] There is no canonical orientation of the ridges, so $\beta \in [0, \pi)$ rather than $[0, 2\pi)$.

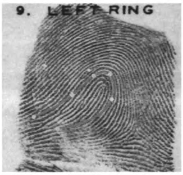


(a) Original

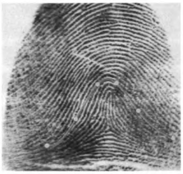


(b) Flow field

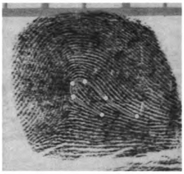
Figure 9.2: Extraction of the flow field from a fingerprint impression. (a) Original (scaled down to 0.3 of its original size, to fit the page). (b) Flow field.



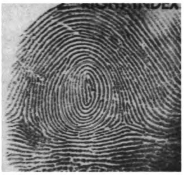
(a) L-loop



(b) Arch



(c) R-loop



(d) Whorl

Figure 9.3: Definitions of fingerprint kernels using splines. Yellow: control points. Red: kernel. The kernels model the shape of the ridge pattern around the center of the fingerprint.

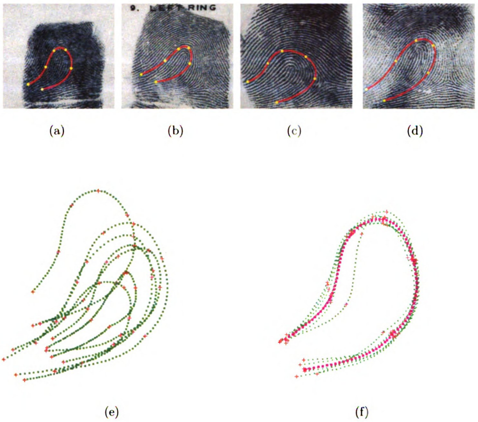


Figure 9.4: Procrustes model for learning fingerprint kernels [78]. (a)-(d) Manually traced left loops in fingerprint images. The yellow dots are the landmarks, homologous across all images. Only four out of 20 fingerprint impressions are shown here. (e) All 20 training contours extracted, unregistered. (f) Registered training contours (green) and their mean (purple).

$$E(\gamma) = \frac{\int_\gamma \sin^2(\alpha - \beta(\gamma))d\gamma}{\int_\gamma d\gamma} \tag{9.1}$$

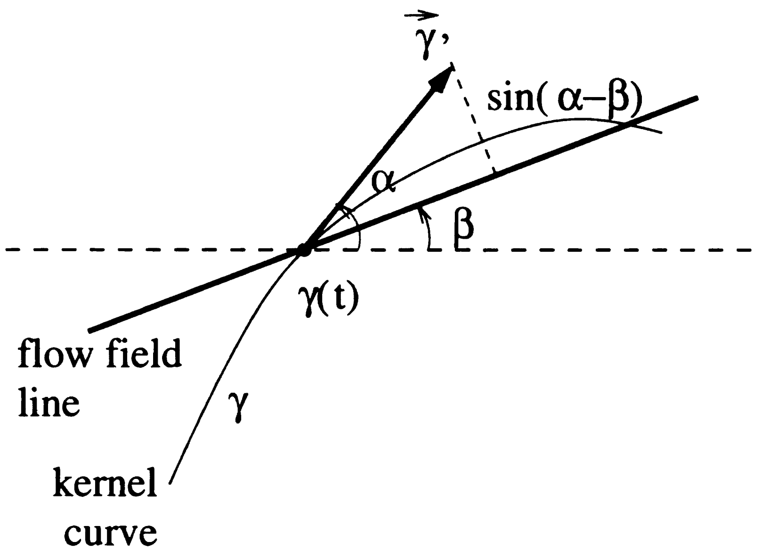


Figure 9.5: The angles α and β defining the measure of fit (Eq. (9.1))

The denominator in Equation (9.1) is a normalization factor and rules out the trivial solution (when γ degenerates to a point).

If γ is allowed to have an arbitrary shape, then the minimum in Equation (9.1) is zero, and it is achieved by any curve γ solution of the ordinary differential system $\dot{\gamma}(t) = V(\gamma(t))$. However, we want to capture patterns of a specific shape, so we apply anisotropic similarity transformations to any of the kernels defined in section 9.2.2, and minimize the energy in Equation (9.1) over these transformations. Specifically, we consider transformations of the form:

$$T(x) = R_\phi \cdot D \cdot x + \tau \qquad \forall x \in \mathbf{R}^2 \tag{9.2}$$

In Equation (9.2), R_ϕ is a rotation of angle ϕ about the origin. $D = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$ is a scaling matrix and $\tau = (\tau_x, \tau_y)$ is a translation vector. If γ_0 is any of the kernels being fit to an image, and $\gamma = T(\gamma_0)$ for any transform T, then the fit energy (Equation (9.1)) becomes simply a five-variable function $E(T) = E(a, b, \phi, \tau_x, \tau_y)$. Minimization of this function is done semi-exhaustively, by testing every quintuple of parameters within an appropriate discrete range, common to all fingerprints in the database. The reason for this is that the density of the ridges produces many local minima, while the absolute minimum is crucial to our method.

To classify a fingerprint impression we fit all four kernels to the image and assign it the type with the best kernel fit. The problem is slightly more complicated, because any fingerprint contains an arch, so an arch kernel will always be a good fit (see Figure 9.8 (c)). The basic idea is then, to consider a fingerprint to be an arch only if no other kernel has a better (or comparable) fit than the arch. Because the non-arch kernels must be fit inside the arch, we term them sub-kernels. For the full details, see [78].

Figure 9.6 shows two impressions of the same finger and the whorl kernel fit to them. The fit is consistent, in the sense that an ellipse of the same size is fit to both impressions, and it is positioned at about the same region in the fingerprints. Note also, that a very good fit is found in (a), even though the kernel curve is not entirely contained in the image. This is the effect of the denominator in the definition of the energy function (Eq. (9.1)).

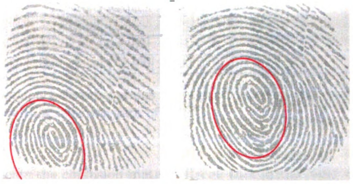

(a) (b)

Figure 9.6: Elliptic kernel fit to two impressions of the same finger.

9.3 Experimental Results with Fingerprints

We used the NIST 4 database [40] to test the proposed method. The database consists of 4000 fingerprint impressions, of size 512 × 512 pixels. Each impression was labeled as one of the "arch", "tented arch", "left loop", "right loop" and "whorl" by a human expert. Some of the images were ambiguous and had more than one label. We considered a fingerprint to be correctly classified if the label assigned by our method was among the labels assigned by the human expert. Furthermore, since in the *natural* distribution of fingerprints the arch and the tented arch combined make up for approximately 5% of all fingerprints [36, 51], we classified both "arch" and "tented arch" simply as "arch". However, we did not alter the distribution of the classes in the NIST database, which is uniform. Our algorithm takes under 0.3 seconds to classify a single image on a Pentium III 800 MHz processor, which yields a total time of under 20 minutes for the entire database of 4000 images. Under these conditions, with no rejection option, we obtained a classification rate of 91.25%. Table 9.1 presents our results in comparison with other classification methods. In the "4 classes" column the arch and tented arch were placed in the same category, whereas in the "5 classes" column, the arch and tented arch were considered distinct.

Algorithm	4 classes	5 classes
Kernel Fitting	91.25%	-
MASK [13]	-	87.1%
Karu & Jain [62]	91.1 %	85.4%
Jain et. al.(*) [60]	94.8%	90%

Table 9.1: Comparison between various fingerprint classification methods. (*) On 2000 fingerprints, with a rejection rate of 1.75%.

Table 9.2 presents the confusion matrix. Recall that in the NIST database some of the fingerprints had two true labels. In building the confusion matrix we counted these images once for each of the true labels. The true labels are along the rows.

We have identified two main reasons for the classification errors that our approach

	L	R	W	A
L	730	2	18	54
R	5	754	7	66
W	32	35	718	11
A	74	39	7	1448

Table 9.2: The confusion matrix.

makes. The first type of error is due to the curve fitting itself: a wrong kernel may have a slightly better score than the true kernel (Figure 9.7). Under the arch - which can be found in any fingerprint (the thick continuous curve near the center of the image) - we fit sub-kernels to determine the class of the fingerprint. The two runners-up are shown under the arch. The two sub-kernels have very close scores, but the wrong type happens by chance to have win by a small margin, and the fingerprint is misclassified.

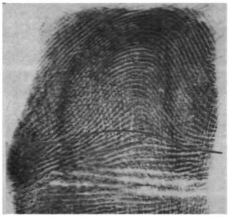
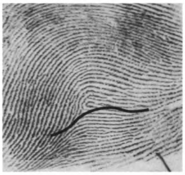
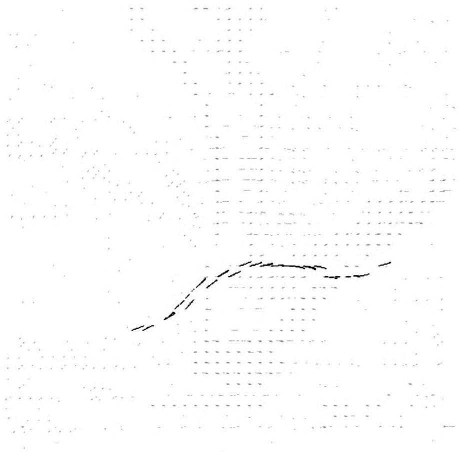


Figure 9.7: Classification error due to wrong best-fit kernel.

The second type of error is not due to the curve fitting process, but rather due to the flow field. In some cases, the fit is good, yet the test sample is misclassified due to the fact that the flow field does not provide a faithful representation of the fingerprint (Figures 9.8 (a) and (b)). This, in turn, is due to the loss of information incurred when the fingerprint is represented as a flow field.



(a)



(b)

Figure 9.8: Sources of classification errors. (a) Right loop fingerprint with an arch as kernel of best fit. (b) The same kernel (dark segments) overlaid on top of the flow field of the impression in (a). The fit is good, but the flow field alone does not show a right loop pattern.

Chapter 10

Conclusions and Future Work

This thesis represents our understanding of fundamental problems in computer vision such as shape learning and shape and border detection.

Central to our work is the very notion of *shape*. This concept is by no means trivial to define, particularly when a mathematical formalism must be developed before any computations can take place. What is "shape"? When should two shapes be considered similar? How do we compare them? Can "shape" be learned? What groups of deformations should be allowed for any given shape? Certainly, these questions have been asked (many times) before, but in our opinion they have not been answered in the most satisfactory or general way. Sometimes simplifying assumptions can be made, such as the existence of homologous points across an entire set of contours (e.g., anatomical landmarks). In all but the simplest cases, point correspondences are not known, or not easy to establish. Even when the 3D objects whose boundaries are being compared do have distinctive landmarks, occlusions create the problem of partial matching, whose naive, brute-force solution has combinatorial complexity.

When point correspondences are not known, matters become significantly more difficult and more confusing. A number of researchers try to recover point correspondences between two curves by warping one curve onto the other. Of course, since any simple closed curve can be deformed in infinitely many ways onto another simple closed curve, one has to choose an optimality criterion in order to single out one particular warping.

Edge detection using active contours has seen a great deal of research, in the past twenty years. It has to be pointed out from the beginning, that the scope of this sort of boundary detector is rather limited, due to the fact that it requires manual initialization. This is the reason why these techniques have had most success in medical image processing, where some amount of human intervention is acceptable and where the space of allowable contours is generally greatly reduced, when compared to natural images.

By and large, existing active contour models are low-level algorithms, i.e., they act at a perceptual level that normally should form the basis for higher level reasoning. These models do not (or cannot!) assume any knowledge about the shape of the object they seek, and invariably are attracted by extraneous distractors.

The point made in this thesis is that statistical shape information can prevent sufficiently *large*, accidental deviations from the true boundaries, and can compensate for incomplete image information, but this must be done with some care. We examine under what circumstances, and to what extent shape memory is beneficial, and we support our conclusions with strong evidence.

Our approach to incorporating a shape prior into the snake evolution is to minimize an energy functional of the form

$$E(\gamma) = E_{img} + E_{int} + E_{shape}$$

where E_{img} quantifies the fit of the snake γ to the image, E_{int} keeps the points on the snake aligned and equally spaced, and E_{shape} is the term that penalizes for deviations from a previously learned shape model.

The experiments suggest, perhaps contrary to intuition, that absolute shape memory (i.e., without regard to the reality of the image data) is detrimental to a boundary-seeking curve, at least for shapes that are insufficiently described by linear transforms, such as human or animal organs. The same experiments show that shape knowledge *is* helpful only when it incorporates some image information as well. This was one unexpected conclusion of our experiments.

These high level conclusions are anchored in the reality of ultrasound images, which was in fact the driving force behind this research. In the process, we came across many practical issues, which turned out to be full blown theoretical problems rather than simple hacks. Most notably, we had to deal with the problem of shape learning without assuming point correspondences, and come up with a model with low computational complexity that could be used online, without significantly slowing down the active contour.

10.1 Contribution

We summarize here some of the technical contributions towards building an active contour model with shape memory, specifically geared toward noisy images.

- We reduce the search space for the minima of the snake energy $E(\gamma)$ from an infinite to a finite dimensional space of functions (curves), by working with *interpolation splines*, rather than with arbitrary curves (Section 6.1 and [77]). Taming the snake energy from a functional to a function of $2k$ variables reduces the inherent difficulties related to its minimization. For one thing, computation of the Euler-Lagrange equations is not necessary, since minimization can be carried out using well understood, well studied optimization routines.

  Furthermore, interpolation splines (as opposed to other types of splines) allow the user to set hard constraints (e.g., keep one or more control points fixed during the snake evolution), because interpolation splines pass through all their control points by definition. This would not be possible, for instance, with B-splines, whose control points *do not* lie on the spline itself.

- We make the image energy term E_{img} depend on the angle between the curve and the image gradient, rather than just the gradient magnitude. This allows control over the polarity of the edge the snake is being attracted to. Specifically, the term

$$E_{img} = -\int_0^1 |\nabla I| dt$$

is replaced with

$$E_{img} = \int_\gamma w(s) \cdot \langle \nabla I^\perp, \frac{\gamma'}{|\gamma'|} \rangle$$

- We drop the original internal energy

$$E_{int} = \int_0^1 |\dot{\gamma}| + |\ddot{\gamma}| ds$$

since the interpolation splines are smooth by construction. We replace it with a term (still denoted by E_{int}) that keeps the points equidistant on the curve by penalizing for deviations from the arclength parametrization:

$$E_{int} = \int_\gamma | \, |\gamma'| - 1| ds$$

This addresses the problem of arclength parametrization effectively and in a principial fashion, quietly treated as "implementation detail" elsewhere.

- Regarding shape learning, we propose a fast and natural alignment method for simple connected curves, that does not assume or require point correspondences.

We term this method *semiaxes alignment* (Section 3.4). For a discretized plane curve γ, we define the semiaxes to be its modes of variation, that is, the eigendirections of the covariance matrix computed on the (finite) set of points resulting from the discretization of γ.

We introduce the notion of *average chamfer transform* (Section 6.2.3), and use it as a shape model, by measuring the simultaneous proximity of a (test) curve, to an entire set of training contours, hence, implicitly, shape similarity.

- We propose an algorithm for segmenting a 1-dimensional piecewise linear signal perturbed by noise, unrelated to active contours (Chapter 7). This algorithm can be used as an edge detector, by applying it along rays starting from the center of an object of interest, but we predict it will have important standalone applications. Currently, the number of constant pieces in the input signal must be specified a priori. We do not have an automatic way of detecting this number, but we suspect that there might be a way to do so by using a minimum description length (MDL) criterion.

- We propose a method for fingerprint classification using again a shape model (Section 9 and [78]) and demonstrate good performance in a first implementation.

10.2 Future Work

The insight we have gained through this research raises more questions than it provides answers. We have some understanding on what shape is, and adding one extra term to the energy functional of a boundary seeking contour is but one way to incorporate the shape memory into the model. But this is by no means the only way, and we believe there may be better ways, which we certainly plan to explore.

First and foremost, we intend to pursue the boundary detector based on the 1-dimensional break-point detection algorithm (Chapter 7), as we believe this approach would have better chances for success. The idea would be to detect the jumps in the intensity profiles along rays starting out from the center of the region of interest, and keep only the strongest, and the most reliable boundary points. We would be left with disparate segments along the target boundaries, which somehow have to be interpolated to encompass the entire object of interest. A shape model would play an important role, and we anticipate having to deal with the problem of partial matching (which parts on the model correspond to the detected segments along the boundaries). We have already done preliminary experiments using this approach (Section 8.5), but more work and a systematic evaluation of the performance is needed.

Secondly, the work on the curvature-based shape representation (Section 2.5) needs to be completed by assessing the discriminatory power of the induced shape similarity metric, in comparison to other known similarity metrics.

There are a number of extensions of the interpolation snakes framework proposed for border detection that can be pursued.

- Apply to other ultrasound problems, specifically left ventricle segmentation.
- Extend to 3D and 2D+time (tracking). Motion video consists of a sequence of 2D frames that are perceived as continuous motion by the human eye when played at a sufficiently high rate. Detecting an object in one of the image and following it in time accross the sequence of frames is the problem of object tracking. When these frames are viewed simply as a stack of 2D images, we are talking about detecting an object in 3-dimensional data. In the case of a video clip, the three dimensions are two spatial dimensions (x and y), with time as the third dimension. One application that doctors would like to have, we are told[1] would be to have the contour of an organ displayed at all times,

[1]According to Dr. Dorry Segev, Division of Transplantation, Department of Surgery, Johns

in real time, during the acquisition of an ultrasound clip as the transducer moves. Needless to say, there are numerous 3D applications in which all three dimensions are spatial. We intend to explore the applicability of our boundary detection methods to 3D data.

- Determine to what extent the globally optimal parameters reported in Tables 8.2, etc. are suboptimal with respect to each individual image.

- Explore the possibility of application of interpolation snakes to detection and tracking of objects (e.g., cars) in real-life images, prompted by the promising (though insufficiently tested) results on the truck images (Section 8.6). Ideally, the system would be fully automatic, which implies that a selective visual attention module and a recognizer be used in conjunction with the border detector.

Hopkins Institutions.

APPENDICES

Appendix A

Landmark registration by linear transforms

A.1 Scaling transforms

Proposition A.1.1 (Scaling registration with known correspondences) *If $T(z) = \lambda z + \tau$ minimizes the squared error energy*

$$\begin{aligned} E(\lambda, \tau) &= \sum_j |T(z_j) - w_j|^2 \\ &= \sum_j (\lambda z_j + \tau - w_j) \cdot \overline{(\lambda z_j + \tau - w_j)}. \end{aligned} \tag{A.1}$$

then its parameters are given by

$$\lambda = \frac{< w', z' >}{|z'|^2} \tag{A.2}$$

$$\tau = -\lambda z_0 + w_0 \tag{A.3}$$

where $< \cdot, \cdot >$ *denotes the complex inner product on* $\mathbb{C}^n$, z_0 *and* w_0 *are the centroids of* z *and* w *respectively, and* $z' = (z_1 - z_0, \ldots z_n - z_0)$ *and* $w' = (w_1 - w_0, \ldots w_n - w_0)$ *are the zero-mean curves.*

Proof: : To minimize this function, one would normally compute the partial derivatives with respect to all four real parameters of the transformation T. Let us recall from calculus, that if $\zeta = x + iy$ is a complex variable, and $\overline{\zeta} = x - iy$ is its conjugate, then one can compute partial derivatives directly with respect to ζ and $\overline{\zeta}$, rather than with respect to x and y. These differential operators are related by

$$\begin{aligned} \partial_\zeta &= \frac{1}{2}(\partial_x - i\partial_y) \\ \partial_{\overline{\zeta}} &= \frac{1}{2}(\partial_x + i\partial_y) \end{aligned}$$

Therefore, in computing the gradient of E, we must compute $\partial_\lambda E$, $\partial_{\overline{\lambda}} E$, $\partial_\tau E$, and $\partial_{\overline{\tau}} E$. However, since for any complex function $f(\zeta)$ we have $\partial_{\overline{\zeta}} \overline{f} = \overline{\partial_\zeta f}$, and because E is a *real* valued function, we only have to compute two (rather than four) sets of derivatives, e.g. $\partial_{\overline{\lambda}} E$ and $\partial_{\overline{\tau}} E$.

$$\partial_{\overline{\lambda}} E = \sum_j (\lambda z_j + \tau - w_j) \cdot \overline{z_j} \tag{A.4}$$

$$\partial_{\overline{\tau}} E = \sum_j (\lambda z_j + \tau - w_j) \tag{A.5}$$

We set these two equations to 0 and solve for τ and λ. First, from Equation (A.5)

$$\tau = -\lambda \cdot z_0 + w_0 \tag{A.6}$$

where $z_0 = (\sum_j z_j)/n$ is the mean of the z_j's and similarly, w_0 is the mean of the

w_j's. This equation can be re-written as $T(z_0) = w_0$, i.e. the optimal transformation maps the center of z onto the center of w.

To find λ, we substitute τ in Equation (A.4) to get

$$\begin{aligned}
\partial_{\overline{\lambda}} E &= \sum_j (\lambda \cdot z_j + \tau - w_j)\overline{z_j} \\
&= \sum_j (\lambda \cdot z_j + \tau - w_j)\overline{(z_j - z_0)} \\
&= \sum_j (\lambda \cdot (z_j - z_0) - (w_j - w_0))\overline{(z_j - z_0)} \\
&= \lambda \sum_j |z'_j|^2 - \sum_j w'_j \overline{z'_j} \\
&= 0
\end{aligned}$$

where $z'_j = z_j - z_0$ and $w'_j = w_j - w_0$. We then get

$$\lambda = \frac{\sum_j w'_j \overline{z'_j}}{\sum_j |z'_j|^2} \tag{A.7}$$

□

A.2 Affine transforms

In this case, there are no constraints on the matrix A:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

In particular, there is no convenient representation of the transform T using complex numbers, so all computations will be carried out over the real numbers.

Let $\overline{X}$ and $\overline{Y}$ represent the means of the X_i's and Y_i's respectively. Let also

$X'_i = X_i - \bar{X} \overset{def}{=} (u'_i, v'_i)$ and $Y'_i = Y_i - \bar{Y} \overset{def}{=} (z'_i, w'_i)$. We collect all the x and y coordinates of the X'_is in two n-dimensional column vectors $U' = (u'_1, u'_2 \dots u'_n)^T$ and $V' = (v'_1, v'_2 \dots v'_n)^T$. Similarly, for the Y'_is we set $Z' = (z'_1, z'_2 \dots z'_n)^T$ and $W' = (w'_1, w'_2 \dots w'_n)^T$.

Proposition A.2.1 (Affine registration with known correspondences) *With the above notations, the coefficients a, b, c, d and τ of the affine transform $T(X) = A \cdot X + \tau$ that minimizes the squared error energy*

$$E(T) = \frac{1}{2} \sum_i |T(\gamma_i) - \delta_i|^2. \tag{A.8}$$

are given by

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} |U'|^2 & \langle U', V' \rangle \\ \langle U', V' \rangle & |V'|^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \langle Z', U' \rangle & \langle W', U' \rangle \\ \langle Z', V' \rangle & \langle W', V' \rangle \end{pmatrix}$$

and $\tau = -(\bar{Y} - A \cdot \bar{X})$.

Proof: In this case, $E = E(a, b, c, d, \tau_x, \tau_y)$. Taking the gradient in Equation (A.8), we get

$$\nabla E = \sum \langle \nabla T(X_i),\ T(X_i) - Y_i \rangle$$

First, by setting the derivatives with respect to τ_x and to τ_y equal to zero, we get, as in the scaling registration case, that $T(\overline{X}) = \overline{Y}$, which is equivalent to $\tau = -(\bar{Y} - A \cdot \bar{X})$. The rest of the proposition follows by setting to zero the derivatives with respect to a, b, c and d.

$\square$

We conclude this appendix with the observation that minimization of the squared error energy (A.8) reduces mathematically to a least squares problem, linear in the coefficients of the unknown transformation T (either scaling or affine). If we identify

T with the column vector formed by its coefficients, then we must solve a system of the form

$$Y = M \cdot T$$

in the least squared sense. This system is nothing but the matrix form of $\nabla E = 0$. If M were a square, invertible matrix, then the solution would be simply $T = M^{-1} \cdot Y$. In general though, the least squares solution is given by $T = M^{\dagger} \cdot Y$ where $M^{\dagger}$ is the Moore-Penrose pseudoinverse of M. Computation of M is usually done using the singular value decomposition (SVD) of M, which, given its popularity in the numerical analysis community, can be used as subroutine. However, the special purpose, closed form formulas for T that we derived here are far more elementary, stable and fast.

Appendix B

Numerics of cubic interpolation splines.

B.1 Representation of a cubic polynomial in terms of second derivatives.

Computation of a cubic interpolation spline is based on the following lemma [1]:

Lemma B.1.1 *If $\gamma = \gamma(t)$ is a cubic polynomial, then on any interval $[a, b]$ γ can be expressed uniquely as*

$$\begin{aligned}\gamma(t) &= A(t) \cdot \gamma(a) + B(t) \cdot \gamma(b) \\ &+ C(t) \cdot \gamma''(a) + D(t) \cdot \gamma''(b)\end{aligned} \tag{B.1}$$

where

$$A = \frac{b-t}{b-a}, \qquad B = 1 - A$$

[1]Mentioned in "Recipes in C" [88] without proof.

$$C = \frac{(A^3 - A)(b-a)^2}{6}, \qquad D = \frac{(B^3 - B)(b-a)^2}{6}$$

Remark B.1.1 *The significance of this lemma is as follows: if we take a, b to be consecutive breakpoints, say, $a = \xi_i$ and $b = \xi_{i+1}$, then, since $\gamma(\xi_i) = y_i$ and $\gamma(\xi_{i+1}) = y_{i+1}$ are control points (hence given), we can compute $\gamma(t)$ for all $t \in [\xi_i, \xi_{i+1}]$, provided we know the second derivatives γ'' at the break points.*

Proof: With A and B as in the statement of the Lemma let us write $\gamma(t) = A\gamma(a) + B\gamma(b) + z(t)$. Clearly,

1. z must be a cubic polynomial (as γ is of degree 3 and A and B are of degree 1).

2. $z(a) = 0$ and $z(b) = 0$, and

3. $z''(a) = \gamma''(a)$ and $z''(b) = \gamma''(b)$.

Since $z(a) = 0$, we shall seek z of the form

$$z(t) = \lambda(t-a)^3 + \mu(t-a)^2 + \nu(t-a)$$

Then the second derivative is

$$z''(t) = 6\lambda(t-a) + 2\mu$$

and by property 3 above, we get

$$\lambda = \frac{\gamma''(b) - \gamma''(a)}{6(b-a)}, \qquad \mu = \frac{\gamma''(a)}{2} \tag{B.2}$$

From $z(b) = 0$, we also get

$$\nu = -(b-a) \cdot \frac{\gamma''(b) + 2\gamma''(a)}{6} \tag{B.3}$$

With these coefficients,

$$\begin{aligned}
z &= \frac{\gamma''(b)-\gamma''(a)}{6(b-a)}(t-a)^3 + \frac{\gamma''(a)}{2}(t-a)^2 - (b-a)\cdot\frac{\gamma''(b)+2\gamma''(a)}{6}(t-a) \\
&= \frac{(b-a)^2}{6}\left[-\frac{(t-a)^3}{(b-a)^3} + 3\frac{(t-a)^2}{(b-a)^2} - 2\frac{(t-a)}{(b-a)}\right]\cdot\gamma''(a) \\
&+ \frac{(b-a)^2}{6}\left[\frac{(t-a)^3}{(b-a)^3} - \frac{(t-a)}{(b-a)}\right]\gamma''(b) \\
&= \frac{(b-a)^2}{6}(-B^3+3B^2-2B)\gamma''(a) + \frac{(b-a)^2}{6}(B^3-B)\gamma''(b) \\
&= \frac{(b-a)^2}{6}(A^3-A)\gamma''(a) + \frac{(b-a)^2}{6}(B^3-B)\gamma''(b)
\end{aligned}$$

□

Corollary B.1.1 *With the same notations as in Lemma B.1.1, the first and second derivatives are*

$$\gamma'(t) = \frac{\gamma(b)-\gamma(a)}{b-a} - \frac{3A^2-1}{6}\cdot(b-a)\cdot\gamma''(a) + \frac{3B^2-1}{6}\cdot(b-a)\cdot\gamma''(b) \quad \text{(B.4)}$$

$$\gamma''(t) = a\cdot\gamma''(a) + b\cdot\gamma''(b) \quad \text{(B.5)}$$

Corollary B.1.2 *With λ, μ and ν as in* (B.2) *and* (B.3), *and with*

$$\tau = \nu + \frac{\gamma(b)-\gamma(a)}{b-a} \quad \text{(B.6)}$$

then on $[a,b]$, the cubic polynomial γ can be written in terms of powers of t as:

$$\gamma(t) = \lambda t^3 + (-3a\lambda+\mu)t^2 + (3a^2\lambda - 2a\mu + \tau)t + (-a^3\lambda + a^2\mu - a\tau + \gamma(a)) \quad \text{(B.7)}$$

Proof:

$$\gamma(t) = \Big(\gamma(a) + \frac{\gamma(b)-\gamma(a)}{b-a}(t-a)\Big) + \underbrace{\lambda(t-a)^3 + \mu(t-a)^2 + \nu(t-a)}_{z(t)}$$

$$= \lambda(t-a)^3 + \mu(t-a)^2 + \underbrace{\Big(\nu + \frac{\gamma(b)-\gamma(a)}{b-a}\Big)}_{\tau}(t-a) + \gamma(a)$$

$$= \lambda t^3 + (-3a\lambda + \mu)t^2 + (3a^2\lambda - 2a\mu + \tau)t + (-a^3\lambda + a^2\mu - a\tau + \gamma(a))$$

□

B.2 Computation of γ_i''

By Remark B.1.1, the second major point is, therefore, the computation of the second derivatives $\gamma''(\xi_j) := \gamma_j''$ at the break points, for $j = 0, \dots l-1$.

This can be done by requiring that the *first* derivatives of γ be continuous *at the break points*, as follows. Consider two consecutive intervals $[\xi_{j-1}, \xi_j]$ and $[\xi_j, \xi_{j+1}]$. The first derivatives of γ on these intervals must agree at ξ_j, so by Corollary B.1.1, after rearranging the terms, we get

$$\Delta\xi_{j-1}\gamma_{j-1}'' + 2(\Delta\xi_{j-1} + \Delta\xi_j)\gamma_j'' + \Delta\xi_j\gamma_{j+1}'' = 6\cdot\Big(\frac{\gamma_{j+1}-\gamma_j}{\Delta\xi_j} - \frac{\gamma_j - \gamma_{j-1}}{\Delta\xi_{j-1}}\Big) \quad \text{(B.8)}$$

where, for brevity, we have denoted $\gamma_j := \gamma(\xi_j)$, $\gamma_j'' := \gamma''(\xi_j)$ and $\Delta\xi_j := \xi_{j+1} - \xi_j$.

In the case of a *closed* spline, equation (B.8) makes sense for $j = 0, \dots l-1$, with the caveat that the indices are read modulo l, and $\xi_l = \xi_{l-1} + |\gamma_0\gamma_{l-1}|$. Therefore, in this case, the matrix of the system is of the form

$$\underbrace{\begin{pmatrix} A_0 & B_0 & 0 & 0 & \dots & 0 & 0 & C_{l-1} \\ C_0 & A_1 & B_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & C_1 & A_2 & B_2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & C_{l-4} & A_{l-3} & B_{l-3} & 0 \\ 0 & 0 & \dots & 0 & 0 & C_{l-3} & A_{l-2} & B_{l-2} \\ B_{l-1} & 0 & \dots & 0 & 0 & 0 & C_{l-2} & A_{l-1} \end{pmatrix}}_{M_c} \cdot \begin{pmatrix} \gamma''_0 \\ \gamma''_1 \\ \gamma''_2 \\ \vdots \\ \gamma''_{l-3} \\ \gamma''_{l-2} \\ \gamma''_{l-1} \end{pmatrix} = \begin{pmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{l-3} \\ D_{l-2} \\ D_{l-1} \end{pmatrix} \tag{B.9}$$

where $C_{j-1} = \Delta\xi_{j-1}$, $B_j = \Delta\xi_j$, $A_j = 2(C_{j-1} + B_j)$ and $D_j = 6 \cdot \left(\frac{\gamma_{j+1}-\gamma_j}{\Delta\xi_j} - \frac{\gamma_j-\gamma_{j-1}}{\Delta\xi_{j-1}}\right)$, for $j = 0, \dots, l-1$. There are as many equations as unknowns $(\gamma''_0, \dots, \gamma''_{l-1})$, so the system is fully determined in the case of a closed spline.

Remark B.2.1 *Note that since $\gamma = (x, y)$ and $\gamma'' = (x'', y'')$, we should in fact write down one version of (B.9) for x and one for y. For brevity though, we're abusing the notation and have γ and γ'' stand in for either x and x'', or for y and y''.*

On the other hand, in the case of an *open* spline, equation (B.8) only makes sense for $j = 1, \dots, l-2$, i.e., only at the interior break points. The system is in this case

$$\underbrace{\begin{pmatrix} A_1 & B_1 & 0 & \dots & 0 & 0 \\ C_1 & A_2 & B_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & C_{l-4} & A_{l-3} & B_{l-3} \\ 0 & \dots & 0 & 0 & C_{l-3} & A_{l-2} \end{pmatrix}}_{M_o} \cdot \begin{pmatrix} \gamma''_0 \\ \gamma''_1 \\ \gamma''_2 \\ \vdots \\ \gamma''_{l-3} \\ \gamma''_{l-2} \\ \gamma''_{l-1} \end{pmatrix} = \begin{pmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{l-3} \\ D_{l-2} \\ D_{l-1} \end{pmatrix} \tag{B.10}$$

Of course, there are in this case $l-2$ equations, two short of the number of unknowns (still $\gamma_0'', \ldots, \gamma_{l-1}''$), so the system (B.10) is *under-determined* as it stands. To make the solution unique, we impose two extra conditions - the **natural end conditions** for the cubic interpolation spline:

$$\gamma_0'' = \gamma_{l-1}'' = 0 \tag{B.11}$$

Introducing (B.11) in (B.10), we get a $(l-2) \times (l-2)$ *triangular* system in the unknowns $\gamma_1'', \ldots, \gamma_{l-2}''$:

$$\underbrace{\begin{pmatrix} A_1 & B_1 & 0 & \ldots & 0 & 0 \\ C_1 & A_2 & B_2 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & C_{l-4} & A_{l-3} & B_{l-3} \\ 0 & \ldots & 0 & 0 & C_{l-3} & A_{l-2} \end{pmatrix}}_{M_O} \cdot \begin{pmatrix} \gamma_1'' \\ \gamma_2'' \\ \vdots \\ \gamma_{l-3}'' \\ \gamma_{l-2}'' \end{pmatrix} = \begin{pmatrix} D_1 \\ D_2 \\ \vdots \\ D_{l-3} \\ D_{l-2} \end{pmatrix} \tag{B.12}$$

A system of the form $M \cdot x = y$ can be solved by an LU decomposition of M [88] in two stages: $L \cdot z = y$ and $M \cdot x = z$. If the matrix M is tri-diagonal, then L and U have very simple forms, and the system can be solved in *linear* time.

Such is the case of M_O, so an open spline can be computed in linear time. In the case of closed splines, however, the matrix M_C is no longer tri-diagonal (but *almost!*) because of B_{l-1} and C_{l-1}. We don't know then, that, a priori, the complexity remains linear. It turns out, after all, that this is still the case, if we compute the *LU* decomposition of M_C. The trick is to *guess* the right form of L and of U.

The following lemma extends the well-known *LU* decomposition for tri-diagonal matrices, to *almost* tri-diagonal matrices, i.e., matrices of the form (B.9).

Lemma B.2.1 *A matrix M of the form (B.9) has an LU decomposition of the form:*

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ c_0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & c_1 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & c_{l-4} & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & c_{l-3} & 1 & 0 \\ u_0 & u_1 & u_2 & \dots & u_{l-4} & u_{l-3} & c_{l-2} & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} a_0 & b_0 & 0 & 0 & \dots & 0 & 0 & 0 & v_0 \\ 0 & a_1 & b_1 & 0 & \dots & 0 & 0 & 0 & v_1 \\ 0 & 0 & a_2 & b_2 & \dots & 0 & 0 & 0 & v_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & a_{l-4} & b_{l-4} & 0 & v_{l-4} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{l-3} & b_{l-3} & v_{l-3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{l-2} & b_{l-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{l-1} \end{pmatrix}$$

Proof: Multiplying L and U above, we get the following relations:

$$a_0 = A_0 \tag{B.13a}$$

$$b_0 = B_0 \tag{B.13b}$$

$$v_0 = C_{l-1} \tag{B.13c}$$

For $k = 0, \dots l-4$,

$$c_k \cdot a_k = C_k \tag{B.14a}$$

$$c_k \cdot b_k + a_{k+1} = A_{k+1} \tag{B.14b}$$

$$b_{k+1} = B_{k+1} \tag{B.14c}$$

$$c_k \cdot v_k + v_{k+1} = 0 \tag{B.14d}$$

For $k = l-3$,

$$c_k \cdot b_k + a_{k+1} = A_{k+1} \tag{B.15a}$$

$$c_k \cdot v_k + b_{k+1} = B_{k+1} \tag{B.15b}$$

Finally, by multiplying the bottom row in L against the columns of M,

$$u_0 \cdot a_0 = B_{l-1} \tag{B.16a}$$

$$u_k \cdot b_k + u_{k+1} \cdot a_{k+1} = 0, \;\; for\; k = 0, \dots l-4 \tag{B.16b}$$

$$u_k \cdot b_k + c_{k+1} \cdot a_{k+1} = C_{k+1}, \;\; for\; k = l-3 \tag{B.16c}$$

$$\sum_{j=0}^{k-3} u_j \cdot v_j + c_{l-2} \cdot b_{l-2} + a_{l-1} = A_{l-1} \tag{B.16d}$$

All unknown entries in L and U, namely $a_0, \dots, a_{l-1}$, $b_0, \dots, b_{l-2}$, $c_0, \dots, c_{l-2}$, as well as $v_0, \dots, v_{l-3}$ and $u_0, \dots, u_{l-3}$ can be computed from equations (B.13), (B.14), (B.15) and (B.16) in *constant* time.

□

We close this appendix with a lemma which relates the LU decomposition of M_O to the LU decomposition of M_C. We expect that since M_O is obtained by deleting the first and last rows and columns from M_C, the LU decomposition of M_O could be obtained in the same manner from the LU decomposition of M_C. This is indeed the case, and it is formalized in the following lemma:

Lemma B.2.2 *Let M_O and M_C be as in (B.10) and (B.9), respectively. Let also $M_O = L_O \cdot U_O$ and $M_C = L_C \cdot U_C$ be their respective LU decompositions. Then L_O and M_O are obtained by deleting the first and last rows and columns of L_C and M_C, respectively.*

Proof: Let L' and M' be the matrices obtained from L_C and M_C, respectively. By Lemma B.2.1,

$$L' = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 \\ c_1 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & c_{l-4} & 1 & 0 \\ 0 & 0 & \dots & 0 & c_{l-3} & 1 \end{pmatrix}$$

$$U' = \begin{pmatrix} a_1 & b_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_2 & b_2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & a_{l-4} & b_{l-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{l-3} & b_{l-3} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{l-2} \end{pmatrix}$$

and let $M' = L' \cdot U'$. We get

$$M'_{11} = a_1, \quad M'_{12} = b_1, \quad M'_{1k} = 0 \quad \forall k = 2, \dots, l-2 \tag{B.17}$$

and for $j \geq 2$

$$M'_{jk} = \begin{cases} c_{j-1}a_k & if\ k = j-1 \\ c_{j-1}b_{k-1} + a_k & if\ k = j \\ b_{k-1} & if\ k = j+1 <= l-2 \\ 0 & otherwise \end{cases} \qquad \forall j = 2 \ldots l-2 \qquad \text{(B.18)}$$

By equations (B.13), (B.14), (B.15) and (B.16) it follows that $M' = 0$ for all but the following entries:

$$M'_{j\,j-1} = C_{j-1}, \quad M'_{jj} = A_j, \quad M'_{j\,j+1} = B_j$$

i.e., $M' = M_O$.

□

This lemma is important when writing the routine to compute the interpolation spline: a *single* routine can handle the LU decomposition in both the open and the closed spline cases. Solving the systems (B.9) and (B.10) is again done by a single routine, and pass it, in addition to the arrays $\{a_j\}$, $\{b_j\}$, $\{c_j\}$, $\{u_j\}$ and $\{v_j\}$, the appropriate range of indices: 0 to $l-1$ for closed splines, and 1 to $l-2$ for open splines.

Bibliography

[1] D. Adalsteinsson and J. A. Sethian. A fast level set method for advancing interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[2] A. A. Amini, T. T. Weymouth, and R. C. Jain. Using dynamic programming for solving variational problems in vision. *PAMI*, 12(9):855–867, September 1990.

[3] D. H. Ballard. Animate vision. *Artificial Intelligence Journal*, 48:57–86, 1991.

[4] H. G. Barrow, J. M. Tennembaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 659–663, Cambridge, MA, 1977.

[5] B. Bascle and R. Deriche. Feature extraction using parametric snakes. In *Proceedings of the 11-th IAPR International Conference on Pattern Recognition*, volume 3, pages 659–663, Netherlands, 1992.

[6] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactios on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002.

[7] P. Besl and N. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[8] F. L. Bookstein. Principal warps: Thin-plate splines and the decomposition of transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567–585, 1989.

[9] G. Borgefors. An improved version of the chamfer matching algorithm. In *Proceedings of the 7th International Conference on Pattern Recognition*, pages 1175–1177, Montreal, Canada, 1984.

[10] G. Borgefors. Hierachical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, November 1988.

[11] P. Brigger, J. Hoeg, and M. Unser. B-spline snakes: A flexible tool for parametric contour detection. *IEEE Transactions on Image Processing*, 9(9):1484–1496, September 2000.

[12] G.T. Candela, P.J. Grother, C.I. Watson, R.A. Wilkinson, and C.L Wilson. Pcasys - a pattern-level clasification automation system for fingerprints. Technical Report NISTIR 5647, NIST, 1995.

[13] R. Capelli, A Lumini, D Maio, and D. Maltoni. Fingerprint classification by directional image partitioning. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(5):402–421, 1999.

[14] V. Caselles, F. Catte, T. Coll, and F. Dibos. A geometric model for active contours. *Numerische Mathematik*, 66:1–31, 1993.

[15] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79, 1997.

[16] T. F. Chan and L. A. Vese. Active contours without edges. *IEEE Transactions of Image Processing*, 10(2):266–277, 2001.

[17] T. Chang. Texture analysis of digitized fingerprints for singularity detection. In *Proc. 5th ICPR*, pages 478–480, 1980.

[18] P. S. Churchland, V. S. Ramachandran, and T. J. Sejnowski. A critique of pure vision. In C. Koch and J. L. Davis, editors, *Large Scale Neuronal Theories of the Brain*, chapter 2, pages 23–60. MIT Press, Cambridge, MA, 1994.

[19] I. Cohen, N. Ayache, and P. Sulger. Tracking points on deformable objects using curvature information. In *ECCV*, pages 458–466, 1992.

[20] L. Cohen. On active contour models and balloons. *CVGIP: Image Understanding*, 53(2):211–218, March 1991.

[21] T. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models - their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, January 1995.

[22] T. F. Cootes and C. J. Taylor. A mixture model for representing shape variation. *Image and Vision Computing*, 17(8):567–574, 1999.

[23] M. G. Crandall and P. L. Lions. Two approximations of solutions to Hamilton-Jacobi equations. *Mathematics of Computation*, 43(167):1–19, July 1984.

[24] D. Cremers, T. Kohlberger, and C. Schnorr. Shape statistics in kernel space for variational image segmentation. *Pattern Recognition*, 36(9):1929–1943, 2003.

[25] D. Cremers, S. Osher, and S. Soatto. Kernel density estimation and intrinsic alignment for knowledge-driven segmentation: Teaching level sets to walk. *Pattern Recognition*, 3175:36–44, September 2004.

[26] C. Davatzikos, J. Prince, and R. Bryan. Image registration based on boundary mapping. *IEEE Proceedings on Transactions on Medical Imaging*, 15:212–215, 1996.

[27] C. de Boor. *A Practical Guide to Splines.* Springer, 2001.

[28] Carl de Boor. B(asic)-spline basics. In Les Piegl, editor, *Fundamental Developments of Computer-Aided Geometric Modeling*, pages 27–49. Academic Press, 1993.

[29] P. Dierckx. *Curve and Surface Fitting with Splines.* Monograhps on Numerical Analysis. Clarendon Press, Oxford, 1995.

[30] I. L. Dryden and K. V. Mardia. *Statistical Shape Analysis.* John Wiley& Sons, Inc., January 1998.

[31] B. A. Dubrovin, A. T. Fomenko, and S. P. Novikov. *Modern Geometry - Methods and Applications*, volume 1. Springer, 1984.

[32] M. P. Dubuisson and A. K. Jain. Modified Hausdorff distance for object matching. In *Proceedings of Int. Conf. on Pattern Recognition*, volume A, pages 566–568, Jerusalem, IS, 1994.

[33] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley, New York, second edition, 2000.

[34] N. Duta. *Learning Based Detection, Segmentation and Matching of Objects.* PhD thesis, Michigan State University, 2000.

[35] N. Duta, A. K. Jain, and M. P. Dubuisson-Jolly. Automatic learning of 2d shape models. *IEEE Transactions on Patern Analysis and Machine Intelligence*, 23(5):433–446, 2001.

[36] FBI. http://www.fbi.gov.

[37] FBI. *Advanced Latent Fingerprint School.* U.S. Department of Justice, Federal Bureau of Investigation, 1983.

[38] FBI. *The Science of Fingerprints.* U.S. Department of Justice, Federal Bureau of Investigation, 1985.

[39] M. Figueredo, J. M. N. Leitao, and A. K. Jain. Unsupervised contour representation and estimation using b-splines and a minimum description length criterion. *IEEE Transactions of Image Processing*, 9(6):1075–1087, June 2000.

[40] U.S. National Institute for Standards and Technology. Nist 4. http://www.itl.nist.gov.

[41] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Boston, second edition, 1990.

[42] D. Geiger, A. Gupta, L. Costa, and J. Vlontzos. Dynamic programming for detecting, tracking and matching deformable contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(3):294–302, 1995.

[43] S. Gold, A. Rangarajan, C. Lu, S. Pappu, and E. Mjolsness. New algorithms for 2d and 3d point matching. *Pattern Recognition*, 31(8):1019–1031, 1998.

[44] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Massachusetts, 1989.

[45] C. Goodall. Procrustes methods in the statistical analysis of shape. *J. Royal Stat. Soc. B*, 53(2):285–339, 1991.

[46] F. Hausdorff. *Grundzüge der Mengenlehre*. Veitu Co., Leipzig, 1914.

[47] F. Hausdorff. *Grundzüge der Mengenlehre*. American Mathematical Society, January 1999.

[48] J. M. Henderson and A. Hollingworth. High-level scene perception. *Annual Review of Psychology*, 50(243-271), 1999.

[49] E. R. Henry. Classification and uses of fingerprints, 1897. See `http://en.wikipedia.org/wiki/Edward_Henry`.

[50] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[51] L. Hong. *Automatic Personal Identification Using Fingerprints*. PhD thesis, Michigan State University, 1998.

[52] B. K. P. Horn. Closed form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4:629–642, 1987.

[53] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour. Closed form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127–1135, 1988.

[54] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[55] L. Itti and C. Koch. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision Research*, 40(10-12):1489–1506, May 2000.

[56] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, Nov 1998.

[57] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[58] A. K. Jain, Y. Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE PAMI.*, 18(3):267–278, 96.

[59] A.K. Jain, L. Hong, and R. Bolle. On-line fingerprint verificatoin. *IEEE Trans. of Pattern Recognition and Machine Intelligence*, 19(4):302–314, 1997.

[60] Anil K. Jain, S. Prabhakar, and Lin Hong. A multichannel approach to fingerprint classification. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(4):348–359, 1999.

[61] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[62] K. Karu and A.K. Jain. Fingerprint classification. *Pattern Recognition*, 29(3):389–404, 1996.

[63] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[64] M. Kawagoe and A. Tojo. Fingerprint pattern classification. *Pattern Recognition*, 17(3):295–303, 1984.

[65] Kitware. The Insight Segmentation and Registration Toolkit (ITK). http://www. itk. org, 2005.

[66] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM J. Optim.*, 9(1):112–147, 1998.

[67] M. F. Land, N. Mennie, and J. Rusted. Eye movements and the roles of vision in activities of daily living. *Perception*, 28:1311–1328, 99.

[68] M. E. Leventon, W. E. L. Grimson, and O. Faugeras. Statistical shape influence in geodesic active contours. In *IEEE Proceedings of CVPR*, volume 1, pages 316–323, Hilton Head Island, North Carolina, June 2000.

[69] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983–1001, 1998.

[70] J. Maintz and M. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.

[71] R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: A level set approach. *PAMI*, 17(2):158–175, February 1995.

[72] D. Marr. *Vision.* Freeman and Co, San Francisco, 1982.

[73] T. McInerney and D. Terzopoulos. Topology adaptive snakes. *Medical Image Analysis*, 4:73–91, 2000.

[74] S. Menet, P. Saint-Marc, and G. G. Medioni. B-snakes: Implementation and application to stereo. In *DARPA90*, pages 720–726, 1990.

[75] S. Minut, J. M. Henderson, R. Falk, F. C. Dyer, and S. Mahadevan. Gaze control for face learning and recognition in humans and machines. In T. Shipley and P. Kellman, editors, *From fragments to objects: Segmentation processes in vision*, pages 463–481. Elsevier, 2001.

[76] S. Minut, S. Mahadevan, J. M. Henderson, and F. C. Dyer. Face recognition using foveal vision. In S-W. Lee, H. H. Bulthoff, and T. Poggio, editors, *Biologically Motivated Computer Vision*, pages 424–433. Springer, 2000.

[77] S. Minut and G. Stockman. Interpolation snakes for border detection in ultrasound images. In A. Ranchordas, H. Araújo, and B. Encarnação, editors, *Proceedings of the First International Conference On Computer Vision and Applications (VISAPP 2006)*, pages 297–305, Setúbal, Portugal, 2006.

[78] Silviu Minut and K. Anil Jain. Hierarchical kernel fitting for fingerprint classification and alignment. In *Proceedings of the International Conference on Pattern Recognition*, Quebec City, Canada, August 2002.

[79] D. Mumford. Mathematical theories of shape: Do they model perception? In *Proc. Conf. 1570 SPIE*, pages 2–10, 1991.

[80] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, XLII:577–685, 1989.

[81] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[82] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[83] S. Osher and J. A. Sethian. Recent numerical algorithms for hypersurfaces moving with curvature-dependent speed: Hamilton-Jacobi equations and conservation laws. *Journal of Differential Geometry, 31, pp. 131–161, 1990*, 31:131–161, 1990.

[84] S. Osher and C. W. Shu. High-order essentially non oscillatory schemes for Hamilton-Jacobi equations. *Siam Journal of Numerical Analysis*, 28(4):907–922, August 1991.

[85] T. Pavlidis. A review of algorithms for shape analysis. *Comp. Vision, Graph. Image Processing*, 7:243–258, 1978.

[86] P. Peona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.

[87] D. Pompeiu. *Sur la continuité des fonctions de variables complexes (Thèse)*. PhD thesis, Ann. Fac. Sci. de Toulouse, 1905.

[88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition edition, 1993. available online at http://www. nr. com.

[89] A. Rangarajan, H. Chui, and F. L. Bookstein. The Softassign procrustes matching algorithm. In *Information Processing in Medical Imaging*, pages 29–42, Berlin, June 1997. Springer-Verlag.

[90] K Rao and K Balk. Type classification of fingerprints: A syntactic approach. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2(3):223–231, 1980.

[91] R. P. N Rao, G. J. Zelinsky, M. M. Hayhoe, and D. H. Ballard. Eye movement in iconic visual search. *Vision Research*, 42(11):1447–1463x, 2002.

[92] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 3 edition, 1986.

[93] D. Rueckert. *Segmentation and Tracking in Cardiovascular MR Images Using Geometrically Deformable Models and Templates*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, London, 1997.

[94] S. Sclaroff and A. Pentland. Modal matching for correspondence and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):545–561, 1995.

[95] T. B. Sebastian, J. J. Crisco, P. N. Klein, and B. B. Kimia. Constructing 2D curve atlases. In *Proceedings of Mathematical Methods in Biomedical Image Analysis*, pages 70–77, 2000.

[96] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci. USA*, 93:1591–1595, February 1996.

[97] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2 edition, 2001.

[98] L. G. Shapiro and G. C. Stockman. *Computer Vision*. Prentice Hall, 2001.

[99] B. F. Skinner. *The Behavior of Organisms*. Appleton-Century-Crofts, New York, 1938.

[100] L. Staib and J. S. Duncan. Boundary finding with parametrically deformable models. *PAMI*, 14(11):1061–1075, November 1992.

[101] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[102] Tagare. Shape-based nonrigid correspondence with application to the heart motion analysis. *IEEE Transactions on Medical Imaging*, 18(7):570–579, 1999.

[103] H. Tagare, D. O'Shea, and A. Rangarajan. A geometric correspondence for shape-based non-rigid correspondence. In *ICCV*, pages 434–439, 1995.

[104] J. Ton and A. K. Jain. Registering Landsat images by point matching. *IEEE Trans. Geosci. Remote Sensing*, 27(5):642–651, 1989.

[105] A. Torralba, A. Oliva, M. S. Castelhano, and J. M. Henderson. Contextual guidance of eye movements and attention in real-world scenes: The role of global features in object search. *Psychological Review*, 113:766–786, 2006.

[106] J. Triesch, D. Ballard, M. Hayhoe, and B. Sullivan. What you see is what you need. *Journal of Vision*, 3:86–94, 2003.

[107] P. Viola and M. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, May 2004.

[108] J. Weng. Developmental robotics: Theory and experiments. *International Journal of Humanoid Robotics*, 1(2):199–236, 2004.

[109] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, Jan 2000.

[110] Donna J. Williams and Mubarak Shah. A fast algorithm for active contours and curvature estimation. *CVGIP: Image Underst.*, 55(1):14–26, 1992.

[111] C. Xu and J. L. Prince. Snakes, shapes and gradient vector flow. *IEEE Transactions on Image Processing*, pages 359–369, March 1998.

[112] C. Y. Xu and J. L. Prince. Gradient vector flow: A new external force for snakes. In *CVPR97*, pages 66–71, 1997.

[113] L. Younes. Computable elastic distance between shapes. *SIAM Journal of Applied Math.*, 58(2):565–586, 1998.

[114] Z. Zhang. Iterative point matching for registration of free-form curves. Technical Report 1658, INRIA-Sophia Antipolis, March 1992.