PROCESSING OF FLAT AND NON-FLAT IMAGE INFORMATION ON ARBITRARY MANIFOLDS USING PARTIAL DIFFERENTIAL EQUATIONS

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

MARCELO BERTALMÍO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
ELECTRICAL ENGINEERING

GUILLERMO SAPIRO, ADVISER

MARCH 2001

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a doctoral thesis by

Marcelo Bertalmío

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

- - - - - - - - - - - - - - - - - - - -

Name of Faculty Adviser

- - - - - - - - - - - - - - - - - - - -

Signature of Faculty Adviser

- - - - - - - - - - - - - - - - - - - -

Date

GRADUATE SCHOOL

Acknowledgements

An unmeasurable amount of thanx goes to Guillermo Sapiro. From $t = 0$ he's been to me the best advisor a human can get, and an excellent friend. He's given me infinite support, freedom, kindness and you definitely wouldn't be reading this if it were not for him.
Another unmeasurable amount of thanx goes to Gregory Randall. His trust, support and hard work allowed this to happen.
Thanx beyond words to Serrana and José Bertalmío.
The work presented in this dissertation was done in cooperation with G. Sapiro, G. Randall, V. Caselles, C. Ballester, S. Osher and L.T. Cheng; many thanx to all of them. Thanx also to S. Betelú, A. Bertozzi, G. Gorla, F. Mémoli, A. Bartesaghi, A. Tannenbaum, University of Minnesota Doctoral Dissertation Fellowships Program, Electrical and Computer Engineering Department (Univ. of MN), Institute Henri Poincare (Paris, France) and Instituto de Ingeniería Eléctrica (Montevideo, Uruguay).

Para Mirta.

Abstract

In this work we have tried to solve several problems that involve scalar and vectorial image information lying on the plane or on 3D surfaces. We will present and discuss four novel algorithms devised to solve those problems: tracking of objects in movies, tracking of regions on deforming 3D surfaces, restoration of damaged pictures, removal of objects from images, and solving of PDE's on implicit surfaces for denoising, texture synthesis and flow visualization. Examples are shown and future lines of research are suggested, both to improve performance and to extend the present techniques to new applications.

Contents

List of Figures

Chapter 1

Introduction

The use of partial differential equations (PDE's) and curvature driven flows in image analysis has become an interest raising research topic in the past few years. The basic idea is to deform a given curve, surface, or image with a PDE, and obtain the desired result as the solution of this PDE. Sometimes, as we will see in the following chapter, a system of coupled PDE's is used. The art behind this technique is in the design and analysis of these PDE's.

Before we comment on the specifics of this work, we shall give a brief introduction on the basics of the field; for this purpose we have heavily drawn from Sapiro's book on the subject, [84]. Then we will present the goals that our work tries to achieve, and in wich way. Finally we will comment on the organization and contents of the following chapters.

1.1 Derivation of PDE's

Partial differential equations can be obtained from variational problems. Assume a variational approach to an image processing problem formulated as

$$\arg\left\{\mathrm{Min}_I \mathcal{U}(u)\right\},$$

where $\mathcal{U}$ is a given energy computed over the image (or surface) I. Let $\mathcal{F}(\Phi)$ denote the Euler derivative (first variation) of $\mathcal{U}$. Since under general assumptions, a necessary condition for I to be a minimizer of $\mathcal{U}$ is that $\mathcal{F}(I) = 0$, the (local) minima may be computed via the steady state solution of the equation

$$\frac{\partial I}{\partial t} = \mathcal{F}(I),$$

where t is an 'artificial' time marching parameter. PDE's obtained in this way have been used already for quite some time in computer vision and image processing, and the literature is large. The most classical example is the Dirichlet integral,

$$\mathcal{U}(I) = \int |\nabla I|^2(x)dx,$$

which is associated with the linear heat equation

$$\frac{\partial I}{\partial t}(t,x) = \Delta I(x).$$

More recently, extensive research is being done on the direct derivation of evolution equations which are not necessarily obtained from the energy approaches. Examples of both types of PDE's are studied in this thesis.

1.2 Advantages of this approach

Clearly, when introducing a new approach to a given research area, one must justify its possible advantages. Using partial differential equations and curve/surface flows in image analysis leads to modeling images in a continuous domain. This simplifies the formalism, which becomes grid-independent and isotropic. The understanding of discrete local nonlinear filters is facilitated when one lets the grid mesh tend to zero and, thanks to an asymptotic expansion, rewrite the discrete filter as a partial differential operator.

Conversely, when the image is represented as a continuous signal, PDE's can be seen as the iteration of local filters with an infinitesimal neighborhood. This interpretation of PDE's allows one to unify and classify a number of the known iterated filters, as well as to derive new ones.

Another important advantage of the PDE approach is the possibility of achieving high speed, accuracy, and stability, with the help of the extensive literature available on numerical analysis. Of course, when considering PDE's for image processing and numerical implementations, we are dealing with derivatives of non-smooth signals, and the right framework must be defined. The theory of *viscosity solutions* provides a framework for rigorously employing a partial differential formalism, in spite of the fact that the image may be not smooth enough to give a classical sense to derivatives involved in the PDE. Last but not least, this area has a quite unique level of formal analysis, giving the possibility to provide not only successful algorithms but also useful theoretical results like existence and uniqueness of solutions.

1.3 When PDE's started to be used for Image Processing

Ideas on the use of PDE's in image processing go back at least to Gabor [35], and a bit more recently, to A. K. Jain [42]. However, the field really took off thanks to the independent works of Koenderink [51] and Witkin [102]. These researchers rigorously introduced the notion of *scale-space*, that is, the representation of images simultaneously at multiple scales. Their seminal contribution is to a large extent the basis of most of the research in PDE's for image processing. In their work, the multi-scale image representation is obtained by Gaussian filtering. This is equivalent to deforming the original image via the classical heat equation, obtaining in this way an isotropic diffusion flow. In the late 80's, R. Hummel [41] noted that the heat flow is not the only parabolic PDE that can be used to create a scale-space, and indeed argued that an evolution equation which satisfies the maximum principle will define a scale-space as well. Maximum principle appears to be a natural mathematical translation of *causality*. Koenderink once again made a major contribution into the PDE's arena when he suggested to add a thresholding operation to the process of Gaussian filtering. As later suggested by Osher and his colleagues, and proved by a number of groups, this leads to a geometric PDE, actually, one of the most famous ones, curvature motion.

Perona and Malik's [77] work on anisotropic diffusion has been one of the most influential papers in the area. They proposed to replace Gaussian smoothing, equivalent to isotropic diffusion via the heat flow, by a selective diffusion that preserves edges. Their work opened a number of theoretical and practical questions that continue to occupy the PDE image processing community, e.g., [2, 81]. In the same framework, the seminal works of Osher and Rudin on shock filters [69] and Rudin, Osher and Fatemi [82] on Total Variation decreasing methods explicitly stated the importance and the need for understanding PDE's for image processing applications.

Many of the PDE's used in image processing and computer vision are based on moving curves and surfaces with curvature based velocities. In this area, the level-set numerical method developed by Osher and Sethian [70] was very influential and crucial. Early developments on this idea were provided in [68], and their equations were first suggested for shape analysis in computer vision in [46]. The basic idea is to represent the deforming curve, surface, or image, as the level-set of a higher dimensional hypersurface. This technique, not only provides more accurate numerical implementations, but also solves topological issues which were very difficult to treat before. The representation of objects as level-sets (zero-sets) is of course not completely new to the computer vision and image processing communities, since it is one of the fundamental techniques in mathematical morphology [83]. Considering the image itself as a collection of its level-sets, and not just as the level-set of a higher dimensional function, is a key concept in the PDE's community [3].

Other works, like the segmentation approach of Mumford and Shah [65] and the snakes of Kass, Witkin, and Terzopoulos have been very influential in the PDE's community as well.

It should be noted that a number of the above approaches rely quite heavily on a large number of mathematical advances in differential geometry for curve evolution [37] and in viscosity solutions theory for curvature motion (see e.g., Evans and Spruck [32].)

The frameworks of PDE's and geometry driven diffusion have been applied to many problems in image processing and computer vision, since the seminal works mentioned above. Examples include continuous mathematical morphology, invariant shape analysis, shape from shading, segmentation, tracking, object detection, optical flow, stereo, image denoising, image sharpening, contrast enhancement, and image quantization.

1.4 Literature on the subject

Important sources of literature are Sapiro's book on PDE's in Image Processing [84], the collection of papers in the book edited by Bart Romeny [81], the book by Guichard and Morel [36] that contains a description of the topic from the point of view of iterated infinitesimal filters, Sethian's book on level-sets [85], Lindeberg's book in Scale-Space theory [55], Weickert's book on anisotropic diffusion in image processing [100], Kimmel's lecture notes [47], Toga's book on Brain Warping that includes a number of PDE's based algorithms for this [93], the special issue on the March 1998 issue of the IEEE Transactions on Image Processing, the special issues in the Journal of Visual Communication and Image Representation, a series of Special Sessions at a number of IEEE International Conference on Image Processing (ICIP), and the Proceedings of the Scale Space Workshop.

1.5 Contribution of this work

In this work we will be dealing with the processing of flat and non-flat image information on arbitrary manifolds using PDE's. It was not our goal to present a "big picture" of the discipline, rather to present and discuss four novel algorithms that we have developed using *PDE-related* techniques. Each of these algorithms was developed as a solution for a particular set of problems.

Specifically, we have tried to solve several (seemingly unrelated) problems that involve scalar and vectorial image information lying on the plane or on 3D surfaces. These problems arised in areas as different as Computer Graphics and Neurobiology. For instance, the object tracking algo-

rithm presented in Chapter 2 was developed for the automatic segmentation of sequences of neuron slices; the *inpainting* algorithm of Chapter 3, developed for image restoration and alteration, can also be used for compression.

The main contribution of our work is that we have created solutions for a given set of problems. These solutions are algorithms that we have designed, implemented and tested. The algorithms perform very well, providing results at least as good as those available in the literature but with a much simpler formulation. This simplicity allows for an easy implementation, involving very little code. No parameter-tuning is required: the algorithms are fast and require little or no user interaction. They have some theory backing them which allows for rigorous improvement, and for the extension to other applications.

1.6 Organization of this work

The book is organized as follows.

Chapter 2 introduces two algorithms that use systems of coupled PDE's and projection of velocities to perform tracking: tracking of objects in movies, tracking of regions on deforming 3D surfaces. The concepts presented here, specially that of projection, will hopefully help to better understand the material to come in chapters 3 and 4.

Chapter 3 presents a novel algorithm for digital inpainting of still images that attempts to replicate the basic techniques used by professional restorators. Inpainting, the technique of modifying an image in an undetectable form, has numerous applications, like the restoration of old photographs and damaged film, the removal of superimposed text like dates, subtitles, or publicity; and the removal of entire objects from the image like microphones or wires in special effects. In contrast with previous approaches, the technique here introduced does not require the user to specify where the missing information comes from. This is automatically done (and in a fast way), thereby allowing to simultaneously fill-in numerous regions containing completely different structures and surrounding backgrounds. In addition, no limitations are imposed on the topology of the region to be inpainted.

Chapter 4 presents a novel framework for solving variational problems and PDE's for scalar and vector-valued data defined on surfaces. The key contribution is that this framework allows working in a fixed Cartesian coordinate system, which eliminates the need for performing complicated and not-accurate computations on triangulated surfaces, as it is commonly done in the graphics and numerical analysis literature. We describe the framework and present examples in texture

synthesis, flow field visualization, as well as image and vector field regularization for data defined on 3D surfaces.

Chapter 5 is the concluding chapter, in which we have tried to briefly summarize the concepts previously introduced, as well as pointing out several lines of future research.

And without further ado, we can now start with the body of the dissertation.

Chapter 2

Preliminary work

In this chapter we will review some work that started during the author's Master's studies but was completed at the Doctoral stage. Two algorithms are introduced: Morphing Active Contours and Region Tracking on Surfaces. Both involve a system of coupled PDE's and use projection of velocities. This preliminary work is introduced both for completeness and as background for the material to come in chapters 3 and 4.

2.1 Morphing Active Contours

A method for deforming curves in a given image to a desired position in a second image is introduced in this section. The algorithm is based on deforming the first image toward the second one via a Partial Differential Equation (PDE), while tracking the deformation of the curves of interest in the first image with an additional, coupled, PDE. The tracking is performed by projecting the velocities of the first equation into the second one. In contrast with previous PDE based approaches, both the images and the curves on the frames/slices of interest are used for tracking. The technique can be applied to object tracking and sequential segmentation. The topology of the deforming curve can change, without any special topology handling procedures added to the scheme. This permits for example the automatic tracking of scenes where, due to occlusions, the topology of the objects of interest changes from frame to frame. In addition, this work introduces the concept of projecting velocities to obtain systems of coupled PDEs for image analysis applications. We show examples for object tracking and segmentation of electronic microscopy.

2.1.1 Introduction

In a large number of applications, we can use information from one or more images to perform some operation on an additional image. Examples of this are given in Figure 2.2. On the top row we have two images of consecutive slices of neural tissue obtained from electronic microscopy. The image on the left has, superimposed, the contour of an object (the boundary of a neuron). We can use this information to drive the segmentation of the next slice, the image on the right. On the bottom row we see two consecutive frames of a video sequence. The image on the left shows a marked object that we want to track. Once again, we can use the image on the left to perform the tracking operation in the image on the right. These are the type of problems we address in this section.

Our approach is based on deforming the contours of interest from the first image toward the desired place in the second one. More specifically, we use a system of coupled PDE's to achieve this (coupled PDE's have already been used in the past to address other image processing tasks, see [81, 80] and references therein). The first PDE deforms the first image, or features of it, toward the second one. The additional PDE is driven by the deformation velocity of the first one, and it deforms the curves of interest in the first image toward the desired position in the second one. This last deformation is implemented using the level-sets numerical scheme developed in [70], allowing for changes in the topology of the deforming curve. That is, if the objects of interest split or merge from the first image to the second one, these topology changes are automatically handled by the algorithm. This means that we will be able to track scenes with dynamic occlusions and to segment 3D medical data where the slices contain cuts with different topologies.

2.1.2 Basic curve evolution

Let $\mathcal{C}(p,t) : I\!R \times [0,\tau) \to I\!R^2$ be a set of closed planar curves. Assume these curves deform "in time" according to

$$\frac{\partial \mathcal{C}(p,t)}{\partial t} = \beta \vec{\mathcal{N}}, \tag{2.1}$$

where β is a given velocity and $\vec{\mathcal{N}}$ the inner unit normal to $\mathcal{C}(p,t)$. We should note that a tangential velocity can be added to the flow, although it will not affect the geometry of the deformation, just the internal parameterization of the curve $\mathcal{C}$. Therefore, (2.1) gives the most general form of geometric deformations for planar curves.

Let's now assume that $\mathcal{C}(p,t)$ is the level-set (isophote) of a given function $u : I\!R^2 \times [0,\tau) \to I\!R$.

Then, in order to represent the evolution of $\mathcal{C}$ by that of u, u must satisfy

$$\frac{\partial u}{\partial t} = \beta \parallel \nabla u \parallel, \tag{2.2}$$

where β is computed at the level-sets of u. This is the formulation introduced by Osher and Sethian [70] to implement curve evolution flows of the type of (2.1). This implementation has several advantages over a direct discretization of (2.1). Probably the main advantage is that changes in the topology of $\mathcal{C}(p,t)$ are automatically handled when evolving u, that is, there is no need for any special tracking of the topology of the level-sets; see [70] for details and [20, 32] for theoretical analysis of this flow. The discretization of (2.2) is performed with an Eulerian approach (fixed coordinate system), as opposed to a Lagrangian approach classically used to discretized (2.1), where marker particles are used. This gives a numerically stable digital-grid implementation. These reasons have motivated the use of this formulation for a large number of applications, including shape from shading, segmentation, mathematical morphology, stereo, and regularization. Extensions of the level-sets algorithm to higher dimensions are of course straightforward.

2.1.3 Morphing active contours

Let $\mathcal{I}_1(x,y,0) : I\!R^2 \rightarrow I\!R$ be the current frame (or slice), where we have already segmented the object of interest. The boundary of this object is given by $\mathcal{C}_{\mathcal{I}_1}(p,0) : I\!R \rightarrow I\!R^2$. Please note that $\mathcal{C}_{\mathcal{I}_1}$ may be *one* curve or *a set* of them, as in most of the examples, so this notation is not entirely correct. Let $\mathcal{I}_2(x,y) : I\!R^2 \rightarrow I\!R$ be the image of the next frame, where we have to detect the new position of the object originally given by $\mathcal{C}_{\mathcal{I}_1}(p,0)$ in $\mathcal{I}_1(x,y,0)$. Let us define a continuous and Lipschitz function $u(x,y,0) : I\!R^2 \rightarrow I\!R$, such that its zero level-set is the curve(s) $\mathcal{C}_{\mathcal{I}_1}(p,0)$. This function can be for example the signed distance function from $\mathcal{C}_{\mathcal{I}_1}(p,0)$. Finally, let's also define $\mathcal{F}_1(x,y,0) : I\!R^2 \rightarrow I\!R$ and $\mathcal{F}_2(x,y) : I\!R^2 \rightarrow I\!R$ to be images representing features of $\mathcal{I}_1(x,y,0)$ and $\mathcal{I}_2(x,y)$ respectively (e.g., $\mathcal{F}_i \equiv \mathcal{I}_i$, or $\mathcal{F}_i$ equals the edge maps of $\mathcal{I}_i$, $i = 1,2$). With these functions as initial conditions, we define the following system of coupled evolution equations (t stands for the marching variable):

$$\begin{aligned} \frac{\partial \mathcal{F}_1(x,y,t)}{\partial t} &= \beta(x,y,t) \parallel \nabla \mathcal{F}_1(x,y,t) \parallel \\ \frac{\partial u(x,y,t)}{\partial t} &= \hat{\beta}(x,y,t) \parallel \nabla u(x,y,t) \parallel \end{aligned} \tag{2.3}$$

where the velocity $\hat{\beta}(x,y,t)$ is given by

$$\hat{\beta}(x,y,t) := \beta(x,y,t)\vec{\mathcal{N}}_{\mathcal{F}_1} \cdot \vec{\mathcal{N}}_u \tag{2.4}$$

and $\vec{\mathcal{N}}_{\mathcal{F}_1}$ and $\vec{\mathcal{N}}_u$ are the normals to the level sets of $\mathcal{F}_1$ and u, respectively:

$$\vec{\mathcal{N}}_{\mathcal{F}_1} := \frac{\nabla \mathcal{F}_1(x,y,t)}{\parallel \nabla \mathcal{F}_1(x,y,t) \parallel}, \;\; \vec{\mathcal{N}}_u := \frac{\nabla u(x,y,t)}{\parallel \nabla u(x,y,t) \parallel}$$

The first equation of this system is the *morphing* equation, where $\beta(x,y,t) : I\!R^2 \times [0,\tau) \rightarrow I\!R$ is a function measuring the 'discrepancy' between the selected features $\mathcal{F}_1(x,y,t)$ and $\mathcal{F}_2(x,y)$. This equation is morphing $\mathcal{F}_1(x,y,t)$ into $\mathcal{F}_2(x,y,t)$, so that $\beta(x,y,\infty) = 0$ and $\mathcal{F}_1(x,y,\infty) = \mathcal{F}_2(x,y,t)$.

The second equation of this system is the *tracking* equation. The velocity in the second equation, $\hat{\beta}$, is just the velocity of the first one ($\beta\vec{\mathcal{N}}_{\mathcal{F}_1}$) projected into the normal direction of the level-sets of u ($\vec{\mathcal{N}}_u$).

Remember that an equation of the type of (2.3) implies that the level sets of $\mathcal{F}_1$ move with normal velocity β and the level sets of u move with normal velocity $\hat{\beta}$. Thus, if we want the zero-level set of u to track the evolution of the points in $\mathcal{F}_1$, the projection of velocities is required: see Figure 2.1. In this figure, the contour $\mathcal{C}$ at time n goes through points A and B in $\mathcal{F}_1$. Point A moves with velocity β normal to its level set in $\mathcal{F}_1$, and at time $n+1$ it has the position A'. Now if we consider A as belonging to $\mathcal{C}$, the zero level set of u, then at time $n+1$ this point has the position A'' in u. Therefore, the curve $\mathcal{C}$ that went through A and B at time n evolves into curve $\mathcal{C}'$ that goes through points A' and B' at time $n+1$. In other words, the projection of velocities allows us to perform tracking.

To conclude, since tangential velocities do not affect the geometry of the evolution, both the level-sets of $\mathcal{F}_1$ and u are following exactly the same geometric flow. In particular, the zero level-set of u is following the deformation of $\mathcal{C}_{\mathcal{I}_1}$, the curve(s) of interest (detected boundaries in $\mathcal{I}_1(x,y,0)$). It is important to note that since $\mathcal{C}_{\mathcal{I}_1}$ is not necessarily a level-set of $\mathcal{I}_1(x,y,0)$ or $\mathcal{F}_1(x,y,0)$, as can be seen in Figure 2.1, u is needed to track the deformation of this curve.

Since the curves of interest in $\mathcal{F}_1$ and the zero level-set of u have the same initial conditions and they move with the same geometric velocity, they will then deform in the same way. Therefore, when the morphing of $\mathcal{F}_1$ into $\mathcal{F}_2$ has been completed, the zero level-set of u should be the curve(s) of interest in the subsequent frame $\mathcal{I}_2(x,y)$.

One could argue that the steady state of (2.3) is not necessarily given by the condition $\beta = 0$, since it can also be achieved with $\parallel \nabla \mathcal{F}_1(x,y,t) \parallel = 0$. This is correct, but it should not affect our tracking since we are assuming that the boundaries to track are not placed over regions where there

is no information and then the gradient is flat. Therefore, for a certain band around our boundaries the evolution will only stop when $\beta = 0$, thus allowing for the tracking operation.

2.1.4 Examples

For the examples in this section, we have opted for a very simple selection of the functions in the tracking system, namely

$$\mathcal{F}_i = \mathcal{L}(\mathcal{I}_i), \quad i = 1, 2, \tag{2.5}$$

and

$$\beta(x, y, t) = \mathcal{F}_2(x, y)) - \mathcal{F}_1(x, y, t), \tag{2.6}$$

where $\mathcal{L}(\cdot)$ indicates a band around $\mathcal{C}_{\mathcal{I}_1}$. That is, for the evolving curve $\mathcal{C}_{\mathcal{I}_1}$ we have an evolving band B of width w around it, and $\mathcal{L}(f(x, y, t)) = f(x, y, t)$ if (x, y) is in B, and it is zero otherwise[1]. This particular *morphing* term in equation (2.6) is a local measure of the difference between $\mathcal{I}_1(t)$ and $\mathcal{I}_2$. It works increasing the grey value of $\mathcal{I}_1(x_0, y_0, t)$ if it is smaller than $\mathcal{I}_2(x_0, y_0)$, and decreasing it otherwise. Therefore, the steady state is obtained when both values are equal $\forall x_0, y_0$ in B, with $\|\nabla \mathcal{I}_1\| \neq 0$. Note that this is a *local* measure, and that no hypothesis concerning the shape of the object to be tracked has been made. Having no model of the boundaries to track, the algorithm becomes very flexible. Being so simple, the main drawback of this particular selection is that it requires an important degree of similarity among the images for the algorithm to track the curves of interest and not to detect spurious objects. If the set of curves $\mathcal{C}_{\mathcal{I}_1}$ isolates an almost uniform interior from an almost uniform exterior as in Figure 2.4, then there is no need for a high similarity among consecutive images. On the other hand, when working with images such as those in Figure 2.3, if $\mathcal{C}_{\mathcal{I}_1}(0)$ is too far away from the expected limit $lim_{t\to\infty}\mathcal{C}_{\mathcal{I}_1}(t)$, then the abovementioned errors in the tracking procedure may occur. This *similarity* requirement concerns not only the shapes of the objects depicted in the image but especially their grey levels, since this β function measures grey-level differences. Therefore, histogram equalization is always performed as a pre-processing operation.

We have found very satisfactory results for these simple selections of features and discrepancy functions. Better selections will of course improve the algorithm performance. We could for example use contrast invariant features to avoid the necessity of histogram equalization (or contrast

[1]The band used for these examples is of three pixels' width, and it gets reinitialized whenever the zero level set of u touches its boundary. The purpose of this band is to reduce errors and computation time

invariant discrepancy functions). We could also use discrepancy functions like correlation or mutual information, or more sophisticated image metrics, to better capture the information present in the band. We should also note that this particular selection of β only involves information of the two present images. Better results are expected if information from additional images in the sequence is taken into account to perform the *morphing* among these two. All these variations are part of the framework here introduced, and the exact selection of the features and discrepancy functions could and should be tailored to the application.

Our *tracking* function u is initialized as the signed distance function to the curve $\mathcal{C}_{I_1}$. Notice that the use of level-set methods allows us to forget about the parameterization of the curve, we work on a fixed grid and just evolve in time the 2D functions $\mathcal{F}_1(x, y, t)$ and $u(x, y, t)$. The numerical implementation is very simple and uses central differences for the computation of the normals as well as *slope limiter* for the norm of the gradients (this is a classical approach, for details see [70]).

The first example of our tracking algorithm is presented in Figure 2.3. This figure shows nine consecutive slices of neural tissue obtained via electronic microscopy (EM). The goal of the biologist is to obtain a three dimensional reconstruction of this neuron. As we observe from these examples, the EM images are very noisy, and the boundaries of the neuron are not easy to identify or to tell apart from other similar objects. Segmenting the neuron is then a difficult task. Before processing for segmentation, the images are regularized using anisotropic diffusion [2, 11, 77]. Since the variation between consecutive slices is not too large, we can use the segmentation obtained for the first slice (segmentation obtained either manually or with the technique described in [98]), to drive the segmentation of the next one, and then automatically proceed to find the segmentation in the following images, recursively using the Morphing Active Contours formulation. Active contours techniques as those in [18, 43, 45, 57] will normally fail with this type of images: 1- The deforming curve gets attracted to local minima, and often fails to detect the neuron; an accurate model of the noise is required. 2- Those algorithms normally deform either inwards or outwards (mainly due to the presence of balloon-type forces), while the boundary curve corresponding to the first image is in general neither inside nor outside the object in the second image. To solve this, more elaborated techniques, e.g., [74, 74], have to be used. Therefore, even if the image is not noisy, special techniques need to be developed and implemented to direct different points of the curve toward different directions.

In Figure 2.3, the top left image shows the manual or semi-automatic segmentation superimposed, while the following ones show the boundaries found by our algorithm. Due to our particular choice of the (simplest) β function, dissimilarities among the images cause the algorithm to mark

as part of the boundary small objects which are too close to our object of interest. These can be removed by simple morphological operations. Cumulative errors might cause the algorithm to lose track of the boundaries after several slices, and re-initialization would be required. Note that these possible problems are simple to correct, while normally active contours techniques will be very far from the solution, making the errors quite impossible to correct.

Figure 2.4 shows an example of object tracking. The top left image has, superimposed, the contours of the objects to track. The following images show the contours found by our algorithm. Notice how topological changes are handled automatically. A pioneering topology independent algorithm for tracking in video sequences, based on the general geodesic framework introduced in [18, 45] can be found in [72]. In contrast with our approach, that scheme is based on a unique PDE, deforming the curve toward a (local) geodesic curve. In [72] more elaborated models to perform tracking are used, and testing on some of the same sequences (e.g., the highway and two-man-walking sequences), we found that a simpler algorithm as the one here proposed already achieves satisfactory results. On the other hand, the Paragios-Deriche technique, including the recently reported extensions [73, 74], are needed for more complicated scenes or longer ones. Note that due to the similarity between frames, our algorithm converges very fast (typically in less than 30 iterations, insuming a few seconds on our PC, of a non-optimal explicit implementation). The CONDENSATION algorithm described in [12] can also achieve, in theory, topology-free tracking, though to the best of our knowledge real examples showing this capability have not been yet reported. In addition, this algorithm requires having a model of the object to track and a model of the possible deformations, even for simple and useful examples as the ones shown in this section (note that the algorithm here proposed requires no previous or learned information). On the other hand, the outstanding tracking capabilities for cluttered scenes shown with the CONDENSATION scheme can not be obtained with the simple selections for $\mathcal{F}_i$ and β used for the examples in this section, and more advanced selections must be investigated. Additional tracking examples are given in the next two figures.

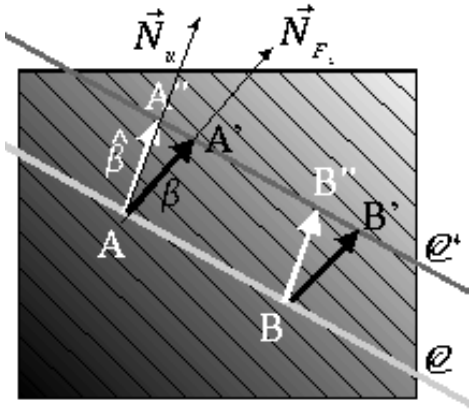

Figure 2.1: Projection of velocities is needed to perform tracking (see text).

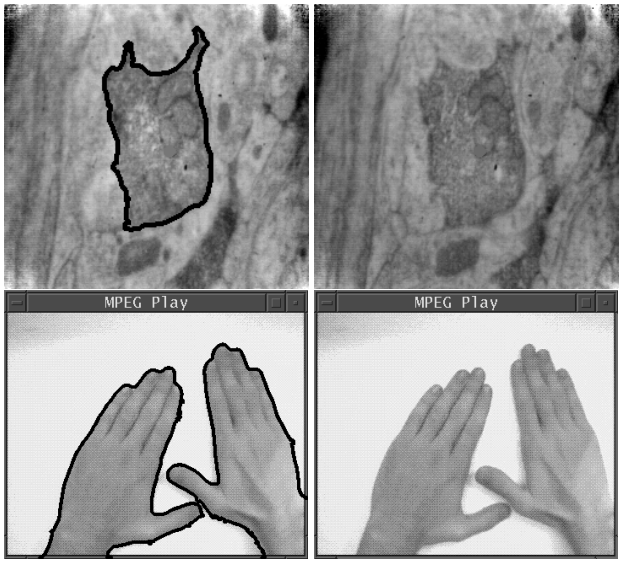

Figure 2.2: Examples of the problems addressed by the Morphing Active Contours algorithm. See text.

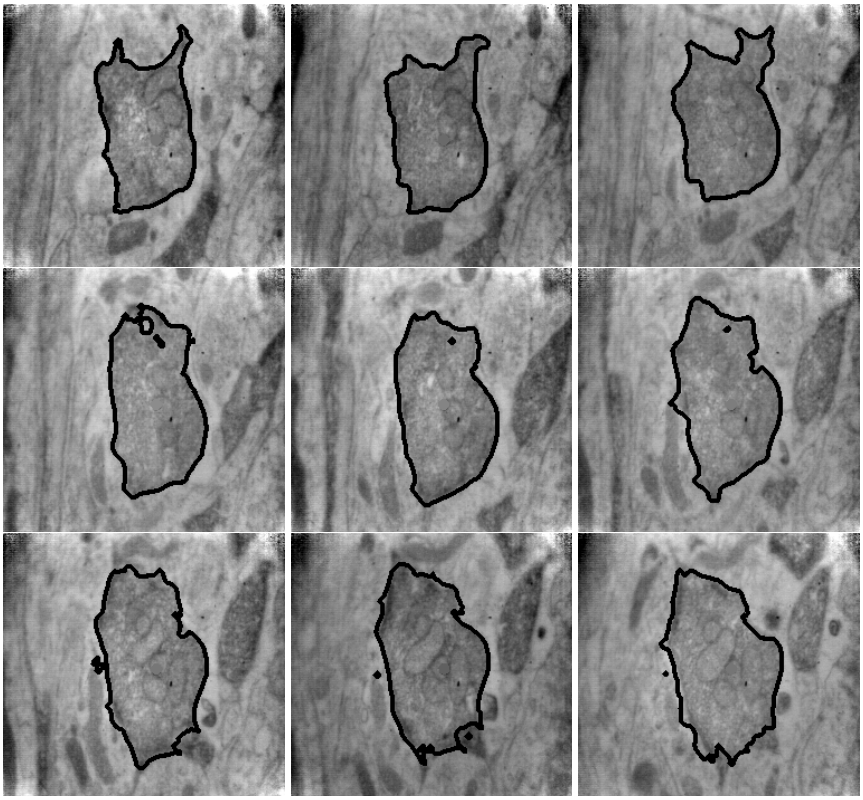

Figure 2.3: Nine consecutive slices of neural tissue. The first image has been segmented manually. The segmentation over the sequence has been performed using the Morphing Active Contours algorithm.

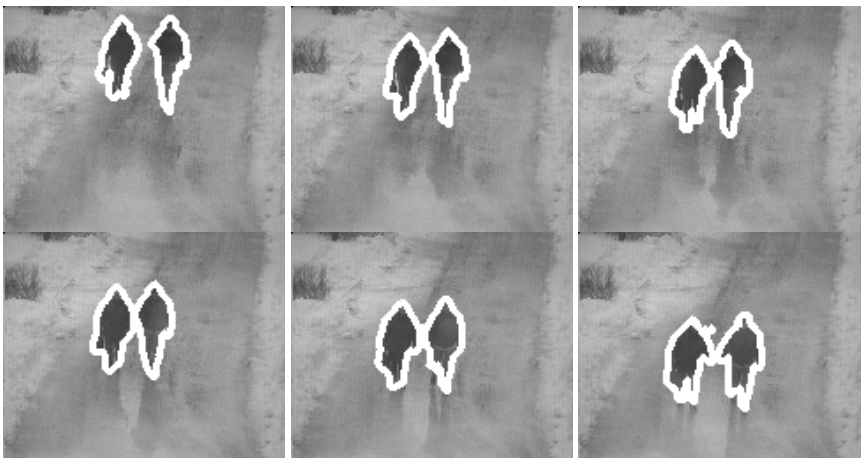

Figure 2.4: Tracking example on the "Walking Swedes" movie.

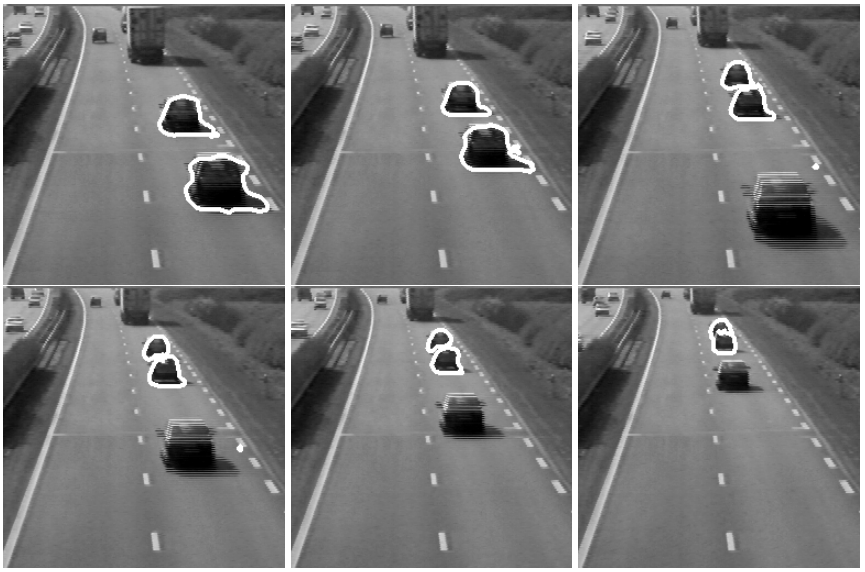

Figure 2.5: Tracking example on the "Highway" movie.

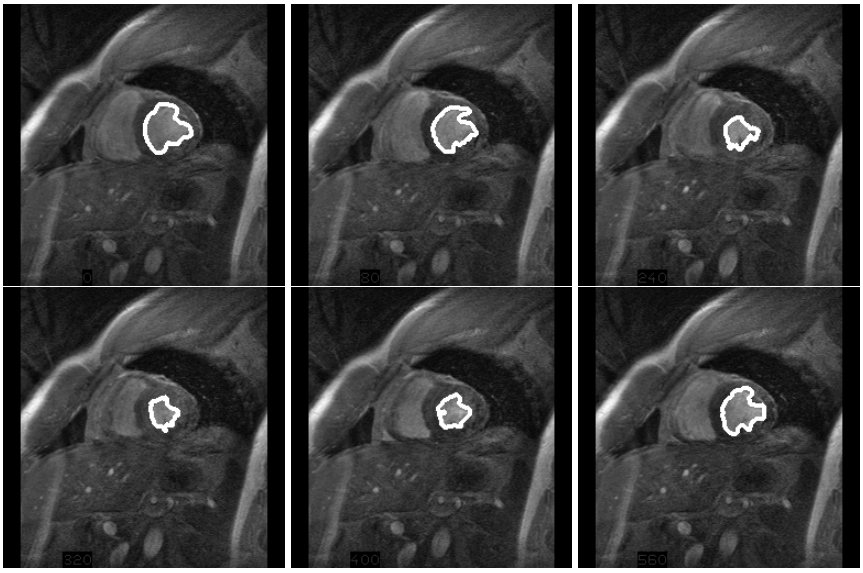

Figure 2.6: Tracking example on the “Heart” movie.

2.2 Region Tracking on Level-Sets Methods

Since the work by Osher and Sethian on level-sets algorithms for numerical shape evolutions, the technique has been used for a large number of applications in numerous fields. In medical imaging, this numerical technique has been successfully used to implement deformable models for segmentation, for example. The migration from a Lagrangian implementation to an Eulerian one via implicit representations or level-sets brought some of the main advantages of the technique, mainly, topology independence and stability. This migration means also that the evolution is parametrization free, and therefore we do not know exactly how each part of the shape is deforming, and the point-wise correspondence is lost. In this note we present a technique to numerically track regions on surfaces that are moving using the level-sets method. The basic idea is to represent the region of interest as the intersection of two implicit surfaces, and then track its deformation from the deformation of these surfaces. This technique then solves one of the main shortcomings of the very useful level-sets approach. Applications include lesion tracking in medical images, region tracking in functional MRI visualization, and geometric surface mapping.

2.2.1 Introduction

The use of level-sets for the numerical implementations of n-dimensional[2] shape deformations became extremely popular following the seminal work of Osher and Sethian [70]

Extending to surfaces the concepts introduced in the previous section for curves, we have that the basic idea is to represent the deformation of a n-dimensional closed surface $\mathcal{S}$ as the deformation of a $n+1$-dimensional function Φ. The surface is represented in an implicit form in Φ, for example, via its zero level-set.

Formally, let's represent the initial surface $\mathcal{S}(0)$ as the zero level-set of Φ, i.e., $\mathcal{S}(0) \equiv \{\mathbf{X} \in I\!R^n \,:\, \Phi(\mathbf{X}, 0) = 0\}$. If the surface is deforming according to

$$\frac{\partial \mathcal{S}(t)}{\partial t} = \beta \vec{\mathcal{N}}_{\mathcal{S}}, \tag{2.7}$$

where $\vec{\mathcal{N}}_{\mathcal{S}}$ is the unit normal to the surface, then this deformation is represented as the zero level-set of $\Phi(\mathbf{X}, t) : I\!R^n \times [0, \tau) \rightarrow I\!R$ deforming according to

$$\frac{\partial \Phi(\mathbf{X}, t)}{\partial t} = \beta(\mathbf{X}, t) \parallel \nabla \Phi(\mathbf{X}, t) \parallel, \tag{2.8}$$

[2] In this note we consider $n \geq 3$.

where $\beta(\mathbf{X}, t)$ is computed on the level-sets of $\Phi(\mathbf{X}, t)$.

In a number of applications, it is important not just to know how the whole surface deforms, but also how some of its regions do. Since the parametrization is missing, this is not possible in a straightforward level-sets approach. This problem is related to the aperture problem in optical flow computation, and it is also the reason why the level-sets approach can only deal with parameterization independent flows that do not contain tangential velocities. Although tangential velocities do not affect the geometry of the deforming shape, they do affect the 'point correspondence' in the deformation. For example, with a straight level-sets approach, it is not possible to determine where a given point $\mathbf{X}_0 \in \mathcal{S}(0)$ is at certain time t. One way to solve this problem is to track isolated points with a set of ODE's, and this was done for example in grid generation; see [85]. This is a possible solution if we are just interested in tracking a number of isolated points. If we want to track regions for example, then using 'particles' brings us back to a 'Lagrangian formulation' and some of the problems that actually motivated the level-sets approach. For example, what happens if the region splits during the deformation? What happens if the region of interest is represented by particles that start to come too close together in some parts of the region and too far from each other in others?

In this note we propose an alternative solution to the problem of region tracking on surface deformations implemented via level-sets.[3] The basic idea is to represent the boundary of the region of interest $\mathcal{R} \in \mathcal{S}$ as the intersection of the given surface $\mathcal{S}$ and an auxiliary surface $\hat{\mathcal{S}}$, both of them given as zero level-sets of $n+1$-dimensional functions Φ and $\hat{\Phi}$ respectively.[4] The tracking of the region $\mathcal{R}$ is given by tracking the intersection of these two surfaces (that is, by the intersection of the level-sets of Φ and $\hat{\Phi}$). In the rest of this section we give details on the technique and present examples.

2.2.2 The algorithm

Assume the deformation of the surface $\mathcal{S}$, given by (2.7), is implemented using the level-sets algorithm, i.e., Equation (2.8). Let $\mathcal{R} \in \mathcal{S}$ be a region we want to track during this deformation, and $\partial\mathcal{R}$ its boundary. Define a new function $\hat{\Phi}(\mathbf{X}, 0) : I\!R^n \to I\!R$ (a distance function for example),

[3] A different level-set approach for mean curvature type of motions of generic 3D curves, together with very deep and elegant theoretical results, is introduced in [4].

[4] The use of multiple level-set functions was used in the past for problems like motion of junctions [61]. Both the problem and its solution are different from the ones in this section.

such that the intersection of its zero level-set $\hat{\mathcal{S}}$ with $\mathcal{S}$ defines $\partial\mathcal{R}$ and then $\mathcal{R}$. In other words,

$$\partial\mathcal{R}(0) := \mathcal{S}(0) \cap \hat{\mathcal{S}}(0) = \{\mathbf{X} \in I\!R^n \,:\, \Phi(\mathbf{X}, 0) = \hat{\Phi}(\mathbf{X}, 0) = 0\}.$$

The tracking of $\mathcal{R}$ is done by simultaneously deforming Φ and $\hat{\Phi}$. The auxiliary function $\hat{\Phi}$ deforms according to

$$\frac{\partial\hat{\Phi}(\mathbf{X}, t)}{\partial t} = \hat{\beta}(\mathbf{X}, t) \parallel \nabla\hat{\Phi}(\mathbf{X}, t) \parallel, \tag{2.9}$$

and then $\hat{\mathcal{S}}$ deforms according to

$$\frac{\partial\hat{\mathcal{S}}}{\partial t} = \hat{\beta}\vec{\mathcal{N}}_{\hat{\mathcal{S}}}. \tag{2.10}$$

We have then to find the velocity $\hat{\beta}$ as a function of β. In order to track the region of interest, $\partial\mathcal{R}$ must have exactly the same geometric velocity both in (2.8) and (2.9). The velocity in (2.8) (or (2.7)) is given by the problem in hand, and is $\beta\vec{\mathcal{N}}_{\mathcal{S}}$. Therefore, the velocity in (2.10) will be the projection of this velocity into the normal direction $\vec{\mathcal{N}}_{\hat{\mathcal{S}}}$ (recall that tangential components of the velocity do not affect the geometry of the flow). That is, for (at least) $\partial\mathcal{R}$,

$$\hat{\beta} = \beta\vec{\mathcal{N}}_{\mathcal{S}} \cdot \vec{\mathcal{N}}_{\hat{\mathcal{S}}}.$$

Outside of the region corresponding to $\mathcal{R}$, the velocity $\hat{\beta}$ can be any function that connects smoothly with the values in $\partial\mathcal{R}$.[5]

This technique, for the moment, requires to find the intersection of the zero-level sets of Φ and $\hat{\Phi}$ at every time step, in order to compute $\hat{\beta}$. To avoid this, we choose a particular extension of $\hat{\beta}$ outside of $\partial\mathcal{R}$, and simple define $\hat{\beta}$ as the projection of $\beta\vec{\mathcal{N}}_{\mathcal{S}}$ for *all* the values of $\mathbf{X}$ in the domain of Φ and $\hat{\Phi}$.[6] Therefore, the auxiliary level-sets flow is given by

$$\begin{aligned}\frac{\partial\hat{\Phi}}{\partial t}(\mathbf{X}, t) &= \left(\beta(\mathbf{X}, t)\,\frac{\nabla\Phi(\mathbf{X}, t)}{\parallel \nabla\Phi(\mathbf{X}, t) \parallel} \cdot \frac{\nabla\hat{\Phi}(\mathbf{X}, t)}{\parallel \nabla\hat{\Phi}(\mathbf{X}, t) \parallel}\right) \\ &\parallel \nabla\hat{\Phi}(\mathbf{X}, t) \parallel,\end{aligned}$$

and the region of interest $\mathcal{R}(t)$ is given by the portion of the zero level-set that belongs to $\Phi(\mathbf{X}, t) \cap \hat{\Phi}(\mathbf{X}, t)$:

$$\partial\mathcal{R}(t) = \{\mathbf{X} \in I\!R^n : \Phi(\mathbf{X}, t) = \hat{\Phi}(\mathbf{X}, t) = 0\}. \tag{2.11}$$

[5]To avoid the creation of spurious intersections during the deformation of Φ and $\hat{\Phi}$, these functions can be re-initialized every few steps, as frequently done in the level-sets approach.

[6]Note that although $\mathcal{S}$ and $\hat{\mathcal{S}}$ do not occupy the same regions in the n dimensional space, their corresponding embedding functions Φ and $\hat{\Phi}$ do have the same domain, making this velocity extension straightforward.

For a number of velocities β, short term existence of the level-sets flow for $\hat{\Phi}$ can be obtained from the results of Evans and Spruck.

This formulation gives the basic region tracking algorithm. In the next subsection, we present some examples.

2.2.3 Examples and comments

We now present examples of the proposed technique. We should note that: (a) Fast techniques like narrow bands, fast-marching [85], or local methods [62], can also be used with the technique here proposed to evolve each one of the surfaces; (b) In the examples below, we compute a zero-order type of intersection between the implicit surfaces, meaning that we consider part of the intersection the full vortex where both surfaces go through (giving a jagged boundary). More accurate intersections can be easily computed using sub-divisions as in marching cubes.

Examples are given in Figure 2.7. In all the columns, the 1st row shows the original surface, and the deformation is shown top to bottom.

Figure 2.7 first shows two toy examples in the first two columns. While the surface moves with a morphing type velocity [6, 7], we are tracking the painted regions on the surfaces. The last row in each column gives a different view of the last step of the deformation. Note how the region of interest changes topology (splits on the left example and merges on the next one).

Figure 2.7, two right columns, presents one of the main applications of this technique. Both these columns first show, on the top, a portion of the human cortex, obtained from MRI and segmented with the technique described in [92]. In order to visualize brain activities recorder via functional MRI in one of the non-visible folds (sulci), it is necessary to 'unfold' the surface, while tracking the color-coded regions. In these examples, the colors simply indicate sign of Gaussian curvature on the original surface (indicating the sulci). We track each one of the colored regions with the technique described in this note, while the surface is deforming with the positive part of Gaussian curvature, and a morphing type velocity [6, 7], respectively. The colors of the deforming surfaces indicate then the sign of the Gaussian curvature in the *original* surface. Note how the surface is unfolded, and the tracking of the colored coded regions allow to find the matching places in the original 3D surface representing the cortex.

The same technique can be applied to visualize lesions that occur on the 'hidden' parts of the cortex. After unfolding, the regions become visible, and the tracking allows to find their position in the original surface. When using level-sets techniques to deform two given shapes, one toward the other (a 3D cortex to a canonical cortex for example), this technique can be used to find the

region-to-region correspondence. This technique then solves one of the basic shortcomings of the very useful level-sets approach.

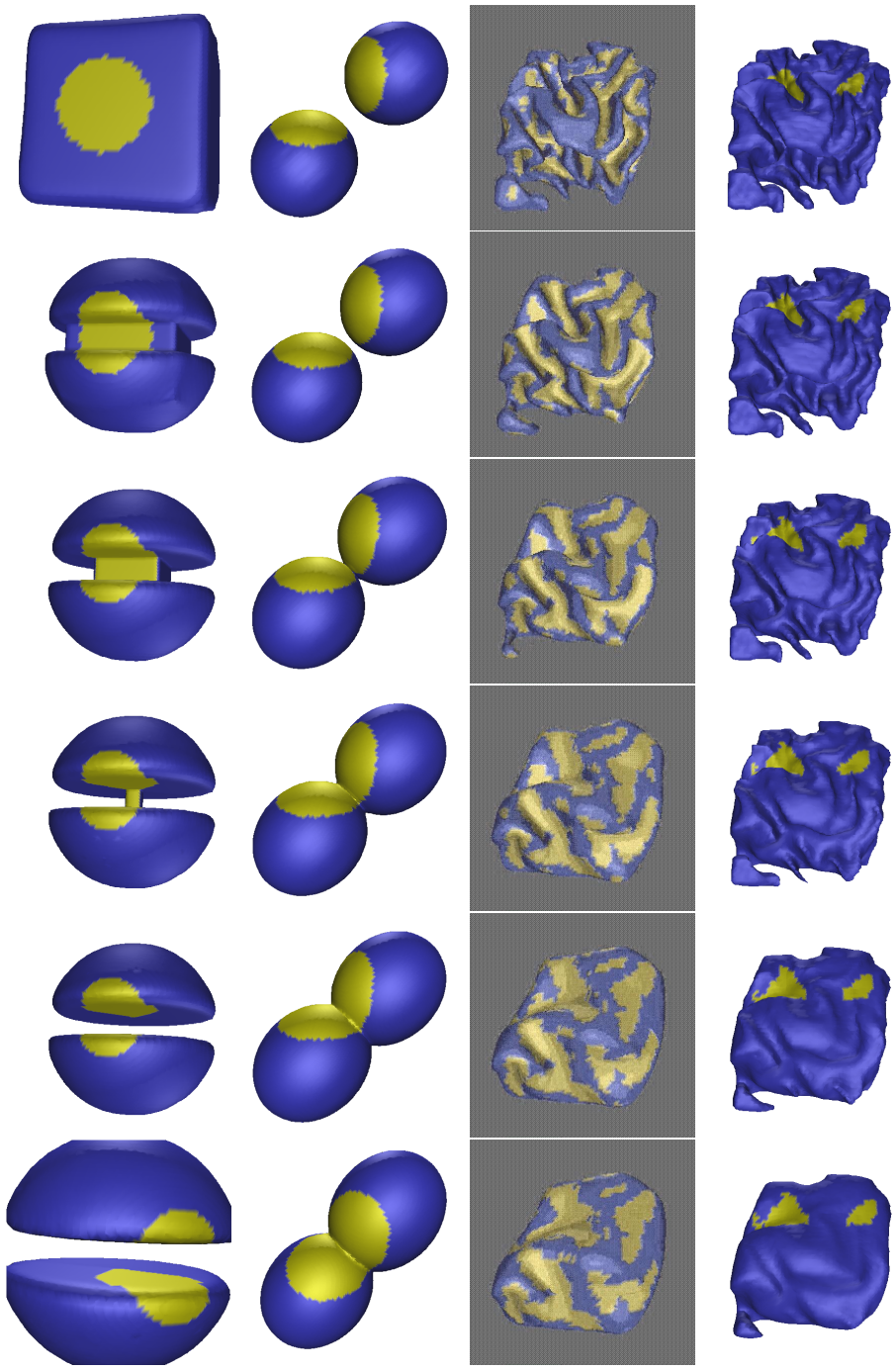

Figure 2.7: Examples of the algorithm introduced in this note (yellow regions are the ones being tracked). The two left columns show toy examples demonstrating possible topological changes on the tracked region. The next columns show unfolding the cortex, and tracking the yellow colored regions, with a Gaussian curvature based flow and a 3D morphing one, respectively.

Chapter 3

Image Inpainting

Inpainting, the technique of modifying an image in an undetectable form, is as ancient as art itself. The goals and applications of inpainting are numerous, from the restoration of damaged paintings and photographs to the removal/replacement of selected objects. In this chapter, we introduce a novel algorithm for digital inpainting of still images that attempts to replicate the basic techniques used by professional restorators. After the user selects the regions to be restored, the algorithm automatically fills-in these regions with information surrounding them. The fill-in is done in such a way that isophote lines arriving at the regions' boundaries are completed inside. In contrast with previous approaches, the technique here introduced does not require the user to specify where the novel information comes from. This is automatically done (and in a fast way), thereby allowing to simultaneously fill-in numerous regions containing completely different structures and surrounding backgrounds. In addition, no limitations are imposed on the topology of the region to be inpainted. Applications of this technique include the restoration of old photographs and damaged film; removal of superimposed text like dates, subtitles, or publicity; and the removal of entire objects from the image like microphones or wires in special effects.

3.1 Introduction

The modification of images in a way that is non-detectable for an observer who does not know the original image is a practice as old as artistic creation itself. Medieval artwork started to be restored as early as the Renaissance, the motives being often as much to bring medieval pictures "up to date" as to fill in any gaps [99, 31]. This practice is called *retouching* or *inpainting*. The object of inpainting is to reconstitute the missing or damaged portions of the work, in order to make it more

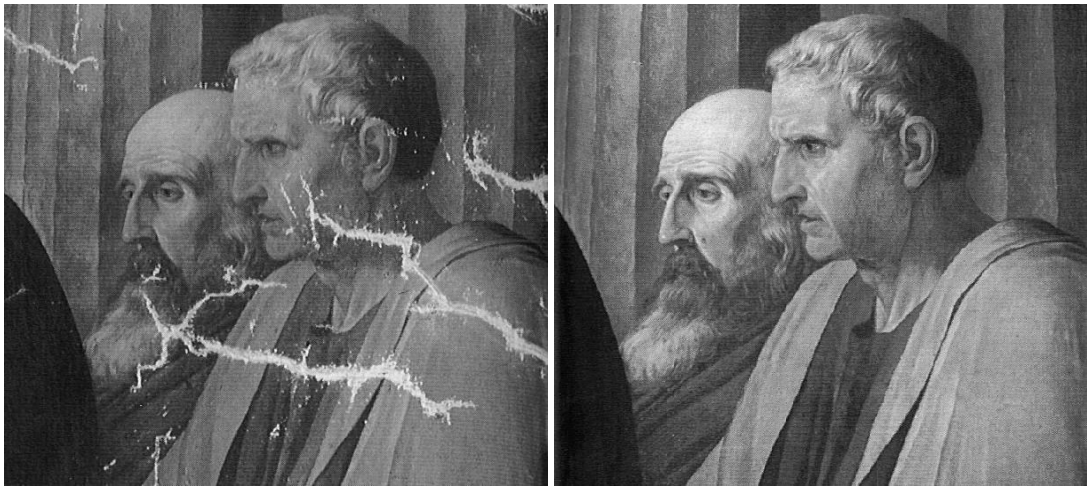

Figure 3.1: Detail of “Cornelia, mother of the Gracchi” by J. Suvee (Louvre). This is a photograph of a manual restoration of a fresco, taken from [31].

legible and to restore its unity [31]. In figure 3.1 we see a detail of an inpainted painting.

The need to retouch the image in an unobtrusive way extended naturally from paintings to photography and film. The purposes remain the same: to revert deterioration (e.g., cracks in photographs or scratches and dust spots in film), or to add or remove elements (e.g., removal of stamped date and red-eye from photographs, the infamous “airbrushing” of political enemies [49]).

Digital techniques are starting to be a widespread way of performing inpainting, ranging from attempts to fully automatic detection and removal of scratches in film [52, 53], all the way to software tools that allow a sophisticated but mostly manual process [14].

In this article we introduce a novel algorithm for automatic digital inpainting, being its main motivation to replicate the basic techniques used by professional restorators. At this point, the only user interaction required by the algorithm here introduced is to mark the regions to be inpainted. Although a number of techniques exist for the semi-automatic detection of image defects (mainly in films), addressing this is out of the scope of this chapter. Moreover, since the inpainting algorithm here presented can be used not just to restore damaged photographs but also to remove undesired objects and writings on the image, the regions to be inpainted must be marked by the user, since they depend on his/her subjective selection. Here we are concerned on how to “fill-in” the regions

to be inpainted, once they have been selected.[1] Marked regions are automatically filled with the structure of their surrounding, in a form that will be explained later in this chapter.

3.2 Related work and our contribution

We should first note that classical image denoising algorithms do not apply to image inpainting. In common image enhancement applications, the pixels contain both information about the real data and the noise (e.g., image plus noise for additive noise), while in image inpainting, there is no significant information in the region to be inpainted. The information is mainly in the regions surrounding the areas to be inpainted. There is then a need to develop specific techniques to address these problems.

Mainly three groups of works can be found in the literature related to digital inpainting. The first one deals with the restoration of films, the second one is related to texture synthesis, and the third one, a significantly less studied class though very influential to the work here presented, is related to disocclusion.

Kokaram et al. [53] use motion estimation and autoregressive models to interpolate losses in films from adjacent frames. The basic idea is to copy into the gap the right pixels from neighboring frames (see figure 3.2). The technique can not be applied to still images or to films where the regions to be inpainted span many frames.

Hirani and Totsuka [39] combine frequency and spatial domain information in order to fill a given region with a selected texture. This is a very simple technique that produces incredibly good results. On the other hand, the algorithm mainly deals with texture synthesis (and not with structured background), and requires the user to select the texture to be copied into the region to be inpainted (see figure 3.3). For images where the region to be replaced covers several different structures, the user would need to go through the tremendous work of segmenting them and searching corresponding replacements throughout the picture. Although part of this search can be done automatically, this is extremely time consuming and requires the non-trivial selection of many critical parameters, e.g., [30]. Other texture synthesis algorithms, e.g., [30, 38, 87], can be used as well to re-create a pre-selected texture to fill-in a (square) region to be inpainted.

In the group of disocclusion algorithms, a pioneering work is described in [67]. The authors

[1] In order to study the robustness of the algorithm here proposed, and not to be too dependent on the marking of the regions to be inpainted, we mark them in a very rough form with any available paintbrush software. Marking these regions in the examples reported in this chapter just takes a few seconds to a non-expert user.

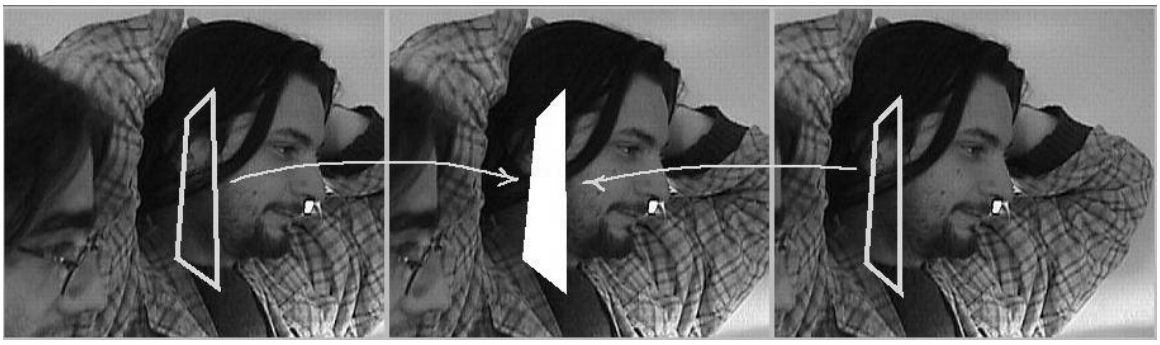

Figure 3.2: Three consecutive frames of a movie are shown. With the technique in [53], the inpainting is performed by using information in adjacent frames.

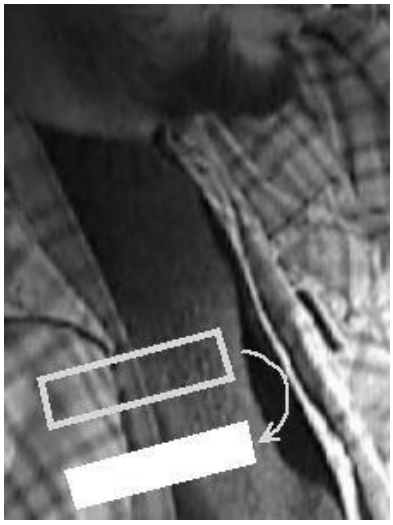

Figure 3.3: With the technique in [39], the user selects what to put where.

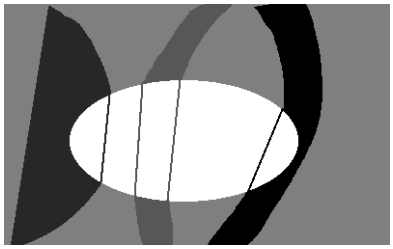

Figure 3.4: With the technique in [59], straight lines are used to join points at the boundary which have equal graylevel.

presented a technique for removing occlusions with the goal of image segmentation.[2] The basic idea is to connect T-junctions at the same gray-level with elastica minimizing curves. The technique was mainly developed for simple images, with only a few objects with constant gray-levels, and will not be applicable for the examples with natural images presented later in this chapter. Masnou and Morel [59] recently extended these ideas, presenting a very inspiring general variational formulation for disocclusion and a particular practical algorithm (not entirely based on PDE's) implementing some of the ideas in this formulation. The algorithm performs inpainting by joining with geodesic curves the points of the isophotes (lines of equal gray values) arriving at the boundary of the region to be inpainted. As reported by the authors, the regions to be inpainted are limited to having simple topology, e.g., holes are not allowed.[3] In addition, the angle with which the level lines arrive at the boundary of the inpainted region is not (well) preserved: the algorithm uses straight lines to join equal gray value pixels (see figure 3.4.) These drawbacks, which will be exemplified later in this chapter, are solved by our algorithm. On the other hand, we should note that this is the closest technique to ours and has motivated in part and inspired our work.

3.2.1 Our contribution

Algorithms devised for film restoration are not appropriate for our application since they normally work on relatively small regions and rely on the existence of information from several frames.

On the other hand, algorithms based on texture synthesis can fill large regions, but require the

[2]Since the region to be inpainted can be considered as occluding objects, removing occlusions is analogous to image inpainting.

[3]This is not intrinsic to the general variational formulation they propose, only to the specific discrete implementation they perform.

user to specify what texture to put where. This is a significant limitation of these approaches, as may be seen in examples presented later in this chapter, where the region to be inpainted is surrounded by hundreds of different backgrounds, some of them being structure and not texture.

The technique we propose does not require any user intervention, once the region to be inpainted has been selected. The algorithm is able to simultaneously fill regions surrounded by different backgrounds, without the user specifying "what to put where." No assumptions on the topology of the region to be inpainted, or on the simplicity of the image, are made. The algorithm is devised for inpainting in structured regions (e.g., regions crossing through boundaries), though it is not devised to reproduce large textured areas. As we will discuss later, the combination of our proposed approach with texture synthesis techniques is the subject of current research.

3.3 The digital inpainting algorithm

3.3.1 Fundamentals

Let Ω stand for the region to be inpainted, and $\partial\Omega$ for its boundary (note once again that no assumption on the topology of Ω is made). Intuitively, the technique we propose will prolong the isophote lines arriving at $\partial\Omega$, while maintaining the angle of "arrival." We proceed drawing from $\partial\Omega$ inward in this way, while curving the prolongation lines progressively to prevent them from crossing each other.

Before presenting the detailed description of this technique, let us analyze how experts inpaint. Conservators at the Minneapolis Institute of Arts were consulted for this work and made it clear to us that inpainting is a very subjective procedure, different for each work of art and for each professional. There is no such thing as "the" way to solve the problem, but the underlying methodology is as follows: (1.) The global picture determines how to fill in the gap, the purpose of inpainting being to restore the unity of the work; (2.) The structure of the area surrounding Ω is continued into the gap, contour lines are drawn via the prolongation of those arriving at $\partial\Omega$; (3.) The different regions inside Ω, as defined by the contour lines, are filled with color, matching those of $\partial\Omega$; and (4.) The small details are painted (e.g. little white spots on an otherwise uniformly blue sky): in other words, "texture" is added.

A number of lessons can immediately be learned from these basic inpainting rules used by professionals. Our algorithm simultaneously, and iteratively, performs the steps (2.) and (3.)

above.[4] We progressively "shrink" the gap Ω by prolonging inward, in a smooth way, the lines arriving at the gap boundary $\partial\Omega$.

3.3.2 The inpainting algorithm

We need to translate the manual inpainting concepts expressed above into a mathematical and algorithmic language. We proceed to do this now, presenting the basic underlying concepts first. The implementation details are given in the next section.

Let $I_0(i,j) : [0,M] \times [0,N] \to I\!R$, with $[0,M] \times [0,N] \subset I\!N \times I\!N$, be a discrete 2D gray level image. From the description of manual inpainting techniques, an iterative algorithm seems a natural choice. The digital inpainting procedure will construct a family of images $I(i,j,n) : [0,M] \times [0,N] \times I\!N \to I\!R$ such that $I(i,j,0) = I_0(i,j)$ and $lim_{n\to\infty} I(i,j,n) = I_R(i,j)$, where $I_R(i,j)$ is the output of the algorithm (inpainted image). Any general algorithm of that form can be written as

$$I^{n+1}(i,j) = I^n(i,j) + \Delta t I_t^n(i,j), \forall (i,j) \in \Omega \tag{3.1}$$

where the superindex n denotes the inpainting "time" n, (i,j) are the pixel coordinates, Δt is the rate of improvement and $I_t^n(i,j)$ stands for the update of the image $I^n(i,j)$. Note that the evolution equation runs only inside Ω, the region to be inpainted.

With this equation, the image $I^{n+1}(i,j)$ is an improved version of $I^n(i,j)$, with the "improvement" given by $I_t^n(i,j)$. As n increases, we achieve a better image. We need now to design the update $I_t^n(i,j)$.

As suggested by manual inpainting techniques, we need to continue the lines arriving at the boundary $\partial\Omega$ of the region Ω to be inpainted (see point (2) in Section 3.3.1). In other words, we need to smoothly propagate information from outside Ω into Ω (points (2) and (3) in Section 3.3.1). Being $L^n(i,j)$ the information that we want to propagate, and $\overrightarrow{N}^n(i,j)$ the propagation direction, this means that we must have

$$I_t^n(i,j) = \overrightarrow{\delta L^n}(i,j) \cdot \overrightarrow{N}^n(i,j), \tag{3.2}$$

where $\overrightarrow{\delta L^n}(i,j)$ is a measure of the change in the information $L^n(i,j)$.[5] With this equation, we estimate the information $L^n(i,j)$ of our image and compute its change along the $\overrightarrow{N}$ direction.

[4] In the discussion section we will argue how both steps can be performed separately, and we will also discuss step (4.).

[5] Borrowing notation from continuous mathematics, we could also write $\overrightarrow{\delta L^n}(i,j)$ as ∇L.

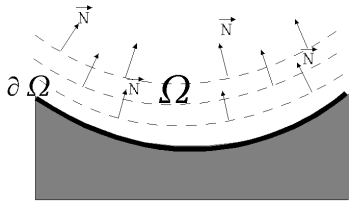

Figure 3.5: Propagation direction as the normal to the signed distance to the boundary of the region to be inpainted.

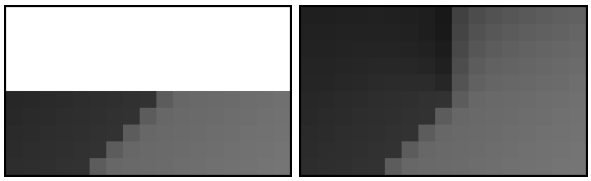

Figure 3.6: Unsuccessful choice of the information propagation direction. Left: detail of the original image, region to be inpainted is in white. Right: restoration.

Note that at steady state, that is, when the algorithm converges, $I^{n+1}(i,j) = I^n(i,j)$ and from (3.1) and (3.2) we have that $\overrightarrow{\delta L^n}(i,j) \cdot \overrightarrow{N}^n(i,j) = 0$, meaning exactly that the information L has been propagated in the direction $\overrightarrow{N}$.

What is left now is to express the information L being propagated and the direction of propagation $\overrightarrow{N}$.

Since we want the propagation to be smooth, $L^n(i,j)$ should be an image smoothness estimator. For this purpose we may use a simple discrete implementation of the Laplacian: $L^n(i,j) := I^n_{xx}(i,j) + I^n_{yy}(i,j)$ (subscripts represent derivatives in this case). Other smoothness estimators might be used, though satisfactory results were already obtained with this very simple selection.

Then, we must compute the change $\overrightarrow{\delta L^n}(i,j)$ of this value along $\overrightarrow{N}$. In order to do this we must first define what the direction $\overrightarrow{N}$ for the 2D information propagation will be. One possibility is to define $\overrightarrow{N}$ as the normal to the signed distance to $\partial\Omega$, i.e., at each point (i,j) in Ω the vector $\overrightarrow{N}(i,j)$ will be normal to the "shrinked version" of $\partial\Omega$ to which (i,j) belongs, see Figure 3.5. This choice is motivated by the belief that a propagation normal to the boundary would lead to the continuity of the isophotes at the boundary. Instead, what happens is that the lines arriving at $\partial\Omega$ curve in order to align with $\overrightarrow{N}$, see Figure 3.6. This is of course not what we expect. Note that the orientation of $\partial\Omega$ is not intrinsic to the image geometry, since the region to be inpainted is arbitrary.

If isophotes tend to align with $\overrightarrow{N}$, the best choice for $\overrightarrow{N}$ is then the isophotes directions. This is a bootstrapping problem: having the isophotes directions inside Ω is equivalent to having the inpainted image itself, since we can easily recover the gray level image from its isophote direction field (see the discussion section and [44]).

We use then a time varying estimation of the isophotes direction field: for any given point (i, j), the discretized gradient vector $\nabla I^n(i, j)$ gives the direction of largest spatial change, while its 90 degrees rotation $\nabla^\perp I^n(i, j)$ is the direction of smallest spatial change, so the vector $\nabla^\perp I^n(i, j)$ gives the isophotes direction. Our field $\overrightarrow{N}$ is then given by the time-varying $\overrightarrow{N}(i, j, n) = \nabla^\perp I^n(i, j)$. We are using a time-varying estimation that is coarse at the beginning but progressively achieves the desired continuity at $\partial\Omega$, instead of a fixed field $\overrightarrow{N}(i, j)$ that would imply to know the directions of the isophotes from the start.

Note that the direction field is not normalized, its norm is the norm of the gradient of $I^n(i, j)$. This choice helps in the numerical stability of the algorithm, and will be discussed in the following subsection.

Since we are performing inpainting along the isophotes, it is irrelevant if $\nabla^\perp I^n(i, j)$ is obtained as a clockwise or counterclockwise rotation of $\nabla I^n(i, j)$. In both cases, the change of $I^n(i, j)$ along those directions should be minimum.

Recapping, we estimate a variation of the smoothness, given by a discretization of the 2D Laplacian in our case, and project this variation into the isophotes direction. This projection is used to update the value of the image inside the region to be inpainted. Writing equation (3.2) in PDE form we get the *inpainting equation*:

$$I_t = \nabla(\Delta I) \cdot \nabla^\perp I \tag{3.3}$$

To ensure a correct evolution of the direction field, a diffusion process is interleaved with the image inpainting process described above.[6] That is, every few steps (see below), we apply a few iterations of image diffusion. This diffusion corresponds to the periodical curving of lines to avoid them from crossing each other, as was mentioned in Section 3.3.1. We use anisotropic diffusion, [77, 2], in order to achieve this goal without losing sharpness in the reconstruction. In particular, we apply a straightforward discretization of the following continuous-time/continuous-space anisotropic diffusion equation:

$$\frac{\partial I}{\partial t}(x, y, t) = g_\epsilon(x, y)\kappa(x, y, t)\left|\nabla I(x, y, t)\right|, \forall (x, y) \in \Omega^\epsilon \tag{3.4}$$

[6] We can also add the diffusion as an additional term in $I_t^n(i, j)$, the results being very similar.

where Ω^ϵ is a dilation of Ω with a ball of radius ϵ, κ is the Euclidean curvature of the isophotes of I and $g_\epsilon(x,y)$ is a smooth function in Ω^ϵ such that $g_\epsilon(x,y) = 0$ in $\partial\Omega^\epsilon$, and $g_\epsilon(x,y) = 1$ in Ω (this is a way to impose Dirichlet boundary conditions for the equation (3.4)).[7]

Let us note a key fact, which was kindly pointed out to us by S. Betelú and A. Bertozzi. The steady state of the inpainting equation (3.3), namely $0 = \nabla(\Delta I) \cdot \nabla^\perp I$, is the same steady state of the so-called Navier-Stokes equation for the stream function of an incompressible inviscid fluid (see for instance [54]). In other words, solving the inpainting problem for an image I is analogous to solving the Navier-Stokes equation (with specific boundary conditions) for a flow with a stream function I. The analogy involves other quantities also: $\nabla^\perp I$ is the velocity, ΔI is the vorticity, etc. The implications of this fact are very important, and will be addressed in Chapter 5.

3.3.3 Discrete scheme and implementation details

The only input to our algorithm are the image to be restored and the mask that delimits the portion to be inpainted. As a preprocessing step, the *whole* original image undergoes anisotropic diffusion smoothing. The purpose of this is to minimize the influence of noise on the estimation of the direction of the isophotes arriving at $\partial\Omega$. After this, the image enters the inpainting loop, where only the values inside Ω are modified. These values change according to the discrete implementation of the inpainting procedure, which we proceed to describe. Every few iterations, a step of anisotropic diffusion is applied (a straightforward, central differences implementation of (3.4) is used; for details see [77, 2]). This process is repeated until a steady state is achieved.

Let $I^n(i,j)$ stand for each one of the image pixels inside the region Ω at the inpainting "time" n. Then, the discrete inpainting equation borrows from the numerical analysis literature and is given by

$$I^{n+1}(i,j) = I^n(i,j) + \Delta t I_t^n(i,j), \forall (i,j) \in \Omega \tag{3.5}$$

where

$$I_t^n(i,j) = \left(\overrightarrow{\delta L^n}(i,j) \cdot \frac{\overrightarrow{N}(i,j,n)}{\left|\overrightarrow{N}(i,j,n)\right|} \right) \left|\nabla I^n(i,j)\right|, \tag{3.6}$$

$$\overrightarrow{\delta L^n}(i,j) := (L^n(i+1,j) - L^n(i-1,j), L^n(i,j+1) - L^n(i,j-1)), \tag{3.7}$$

[7] Other filters, e.g., form mathematical morphology, can be applied as well, though we found the results obtained with this equation satisfactory.

$$L^n(i,j) = I^n_{xx}(i,j) + I^n_{yy}(i,j), \tag{3.8}$$

$$\frac{\overrightarrow{N}(i,j,n)}{\left|\overrightarrow{N}(i,j,n)\right|} := \frac{(-I^n_y(i,j), I^n_x(i,j))}{\sqrt{(I^n_x(i,j))^2 + (I^n_y(i,j))^2}}, \tag{3.9}$$

$$\beta^n(i,j) = \overrightarrow{\delta L^n}(i,j) \cdot \frac{\overrightarrow{N}(i,j,n)}{\left|\overrightarrow{N}(i,j,n)\right|}, \tag{3.10}$$

and

$$|\nabla I^n(i,j)| = \begin{cases} \sqrt{(I^n_{xbm})^2 + (I^n_{xfM})^2 + (I^n_{ybm})^2 + (I^n_{yfM})^2}, \\ \text{when } \beta^n > 0 \\ \sqrt{(I^n_{xbM})^2 + (I^n_{xfm})^2 + (I^n_{ybM})^2 + (I^n_{yfm})^2}, \\ \text{when } \beta^n < 0 \end{cases} \tag{3.11}$$

We first compute the 2D smoothness estimation L in (3.8) and the isophote direction $\overrightarrow{N}/\left|\overrightarrow{N}\right|$ in (3.9). Then in (3.10) we compute β^n, the projection of $\overrightarrow{\delta L}$ onto the (normalized) vector $\overrightarrow{N}$, that is, we compute the change of L along the direction of $\overrightarrow{N}$. Finally, we multiply β^n by a *slope-limited* version of the norm of the gradient of the image, $|\nabla I|$, in (3.11). [8] A central differences realization would turn the scheme unstable, and that is the reason for using slope-limiters. The subindexes b and f denote backward and forward differences respectively, while the subindexes m and M denote the minimum or maximum, respectively, between the derivative and zero (we have omitted the space coordinates (i,j) for simplicity); see [70] for details. A pseudo-code for the numerical implementation of the inpainting equation is included in Appendix A. Finally, let us note that the choice of a non-normalized field $\overrightarrow{N}$ instead of a normalized version of it allows for a simpler and more stable numerical scheme; see [58, 82].

Note once again that when the inpainting algorithm arrives to steady state, that is, $I_t = 0$, we have geometrically solved $\nabla(\text{Smoothness}) \cdot \nabla^{\perp} I = 0$, meaning that the "smoothness" is constant along the isophotes. [9]

When applying equations (3.5)-(3.11) to the pixels in the border $\partial\Omega$ of the region Ω to be inpainted, known pixels from outside this region are used. That is, conceptually, we compute

[8] Note that $|\nabla I| = |\nabla^{\perp} I|$.

[9] This type of information propagation is related and might be applicable to velocity fields extension in level-set techniques [71, 109].

equations (3.5)-(3.11) in the region Ω^ϵ (an ϵ dilation of Ω), although we update the values only inside Ω (that is, (3.5) is applied only inside Ω). The information in the narrow band $\Omega^\epsilon - \Omega$ is propagated inside Ω. Propagation of this information, both gray-values and isophotes directions, is fundamental for the success of the algorithm.

In the restoration loop we perform A steps of inpainting with (3.5), then B steps of diffusion with (3.4), again A steps of (3.5), and so on. The total number of steps is T. This number may be pre-established, or the algorithm may stop when changes in the image are below a given threshold. The values we use are: $A = 15, B = 2$, at speed $\Delta t = 0.1$. The value of T depends on the size of Ω. If Ω is of considerable size, a *multiresolution* approach is used to speed-up the process: we basically use the converged result of a lower resolution stage to initialize the higher one, as it is classically done in image processing. I.e., we take the original image I, reduce its dimensions to, say, 25% obtaining $I_{0.25}$, perform its inpainting obtaining as a result $Itmp_{0.25}$, then increase the size of $Itmp_{0.25}$ by a 100% to $Itmp_{0.50}$, use this as an initial condition for the next inpainting operation, and so on.

Color images are considered as a set of three images, and the above described technique is applied independently to each one. For instance, if the color image to inpaint I uses the traditional (R, G, B) color model, we can decompose it into three scalar ("grayscale") images I_R, I_G and I_B; we perform the inpainting on each of the three images separately and then obtain an inpainted color image by combining the resulting inpaintings of I_R, I_G and I_B.

To avoid the appearance of spurious colors, we use a color model (very similar to the LUV model) with one luminance (ρ) and two chroma ($sin\phi$,$sin\psi$) components. See Figure 3.7. From the tradtional (R, G, B) color model we can get to ours with these simple formulas: $\rho = \sqrt{R^2 + G^2 + B^2}$, $sin\psi = \frac{G}{\rho}$, and $sin\phi = \frac{R}{\sqrt{R^2+B^2}}$. Conversely, to go back to a (R, G, B) image we use: $G = \rho sin\psi$, $R = \rho\sqrt{1-\rho^2}sin\phi$, $B = \sqrt{\rho^2 - G^2 - R^2}$.

3.4 Results

The CPU time required for inpainting depends on the size of Ω. In all the color examples here presented, the inpainting process was completed in less than 5 minutes (for the three color planes), using non-optimized C++ code running on a PentiumII PC (128Mb RAM, 300MHz) under Linux. All the examples use images available from public databases over the Internet. The main examples here presented, and additional ones, can be seen at http://www.ece.umn.edu/users/marcelo/restoration.html, where in addition to the original and inpainted images reproduced below, the evolution process can

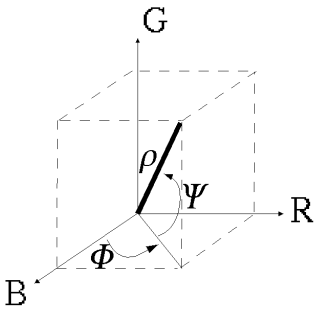

Figure 3.7: Relation between the (R, G, B) color model and the one used in this article, $(\rho, sin\phi, sin\psi)$.

be observed.

Figure 3.8 shows, on the left, a synthetic image with the region to inpaint in white. Here Ω is large (30 pixels in diameter) and contains a hole. The inpainted reconstruction is shown on the right. Notice that contours are recovered, joining points from the inner and outer boundaries. Also, these reconstructed contours follow smoothly the direction of the isophotes arriving at $\partial\Omega$ (the algorithm reported in [59] will fail with this type of data).

Figure 3.9 is a toy example that shows that the choice of a clockwise (CW) or counterclockwise (CCW) rotation of the gradient does not affect the final result. The top row shows the evolution for a CW rotation, the bottom row the evolution for a CCW rotation. Eventhough the intermediate steps are different, the steady state achieved is the same.

Figure 3.10 shows the influence of the diffusion steps in our algorithm. If too little diffusion is used, shocks are developed and the evolution turns unstable. With the number of steps suggested previously, the result is good. With too much diffusion, the result is poor: information is blurred inside Ω, does not propagate from outside.

Figure 3.11 shows a deteriorated B&W image (first row) and its reconstruction (second row). As in all the examples in this article, the user only supplied the "mask" image (last row). This mask was drawn manually, using a paintbrush-like program. The variables were set to the values specified in the previous section, and the number of iterations T was set to 3000. When multiresolution is not used, the CPU time required by the inpainting procedure was approximately 7 minutes. With a 2-level multiresolution scheme, only 2 minutes were needed. Observe that details in the nose and right eye of the middle girl could not be completely restored. This is in part due to the fact that the mask covers most of the relevant information, and there is not much to be done without the use of

high level prior information (e.g., the fact that it is an eye). These minor errors can be corrected by the manual procedures mentioned in the introduction, and still the overall inpainting time would be reduced by orders of magnitude. This example was tested and showed to be robust to initial conditions inside the region to be inpainted.

Figures 3.12 and 3.13 show two vandalized color images and their corresponding restorations, followed by an example where overimposed text is removed from the image (figure 3.13.) These are typical examples where texture synthesis algorithms as those described in the introduction can not be used, since the number of different regions to be filled-in is very large.

The next figure shows the progressive nature of the algorithm, several intermediate steps of the inpainting procedure are shown, removing painted text over a natural scene.

Finally, Figure 3.16 shows an entertainment application. The bungee cord and the knot tying the man's legs have been removed. Given the size of Ω a 2-level multiresolution scheme was used. Here it becomes apparent that it is the user who has to supply the algorithm with the masking image, since the choice of the region to inpaint is completely subjective.

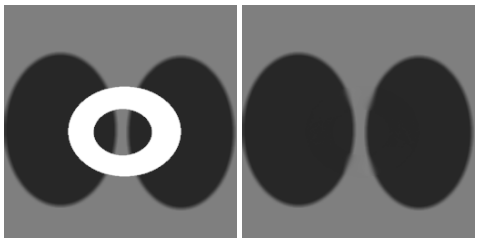

Figure 3.8: Synthetic example: Ω is shown in white. Topology is not an issue, and the recovered contours smoothly continue the isophotes.

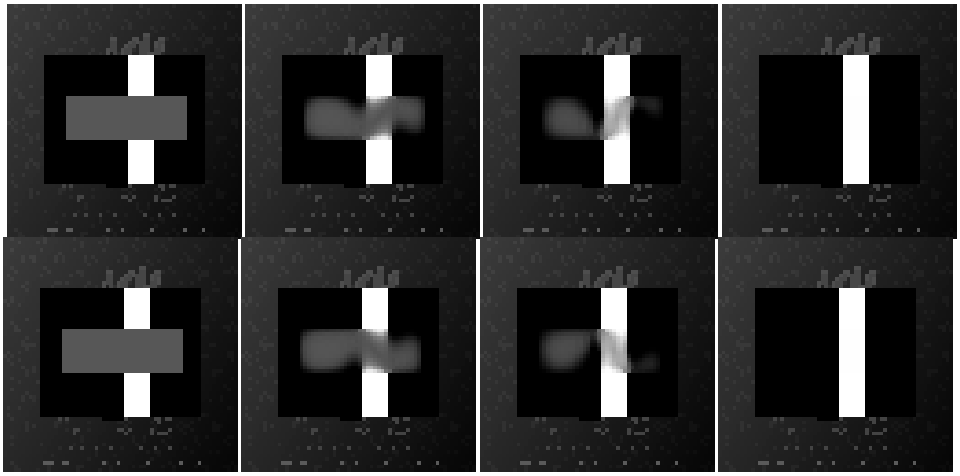

Figure 3.9: Top: evolution using CW rotation of gradient. Bottom: evolution using CCW rotation of gradient. Note that the steady state is the same.

Figure 3.10: From left to right: original image, result with no diffusion, result with some diffusion, result with too much diffusion.

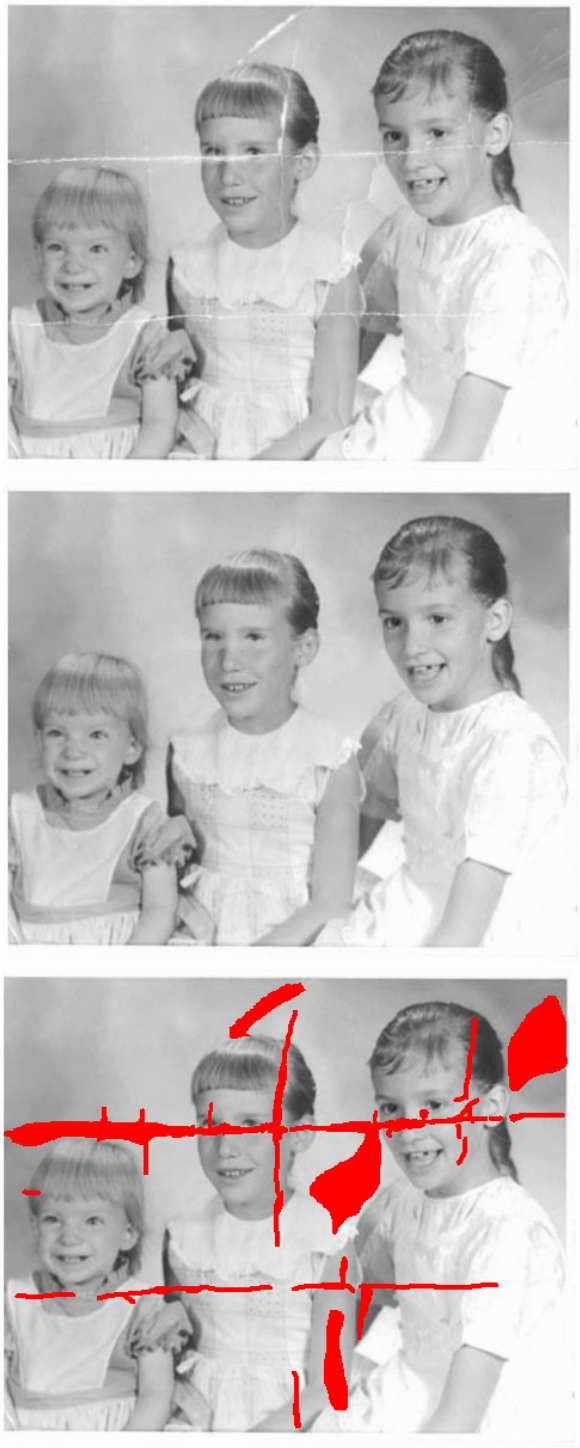

Figure 3.11: Restoration of an old photograph.

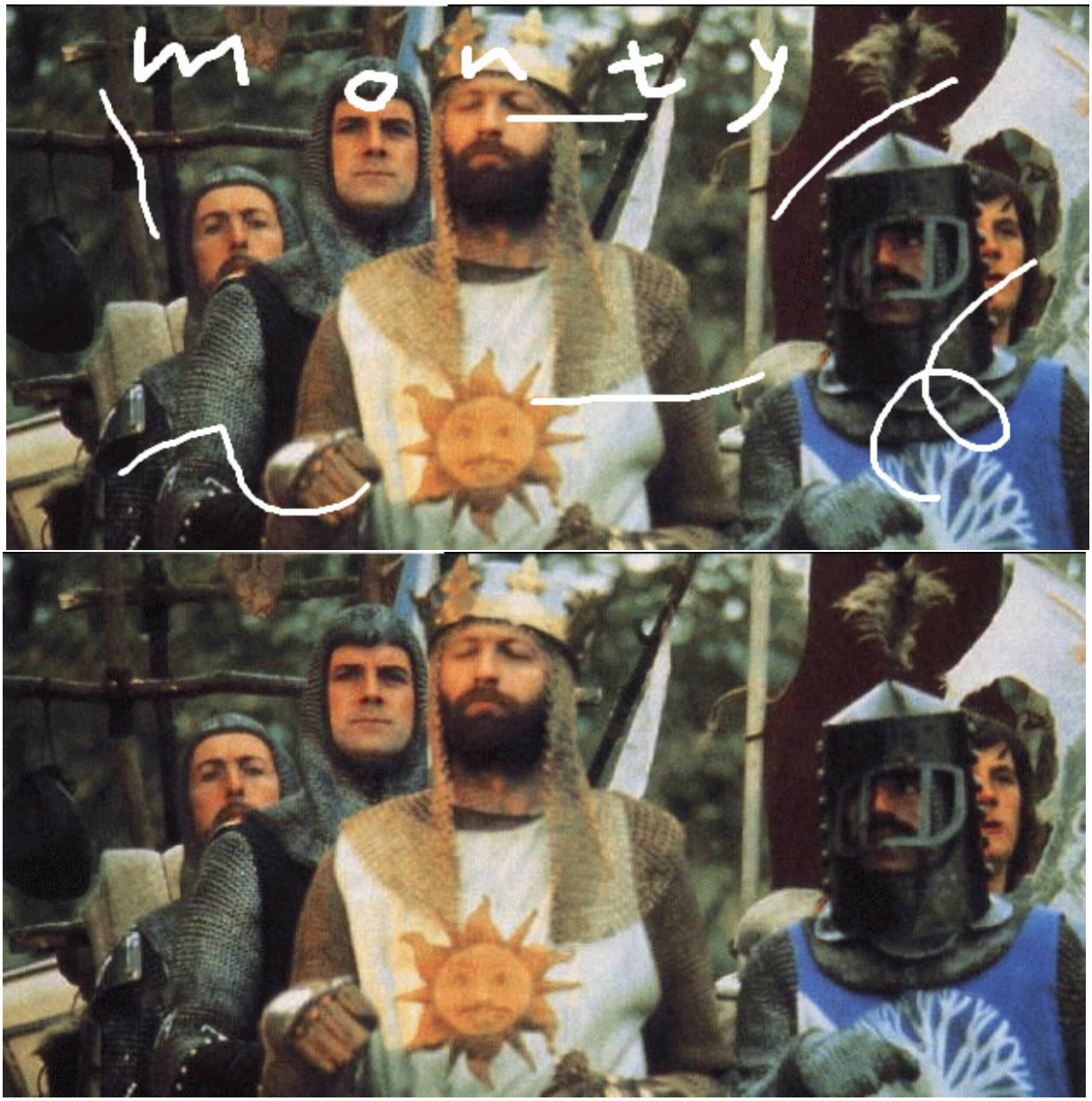

Figure 3.12: Restoration of color images.

Figure 3.13: Restoration of color images (cont'd).

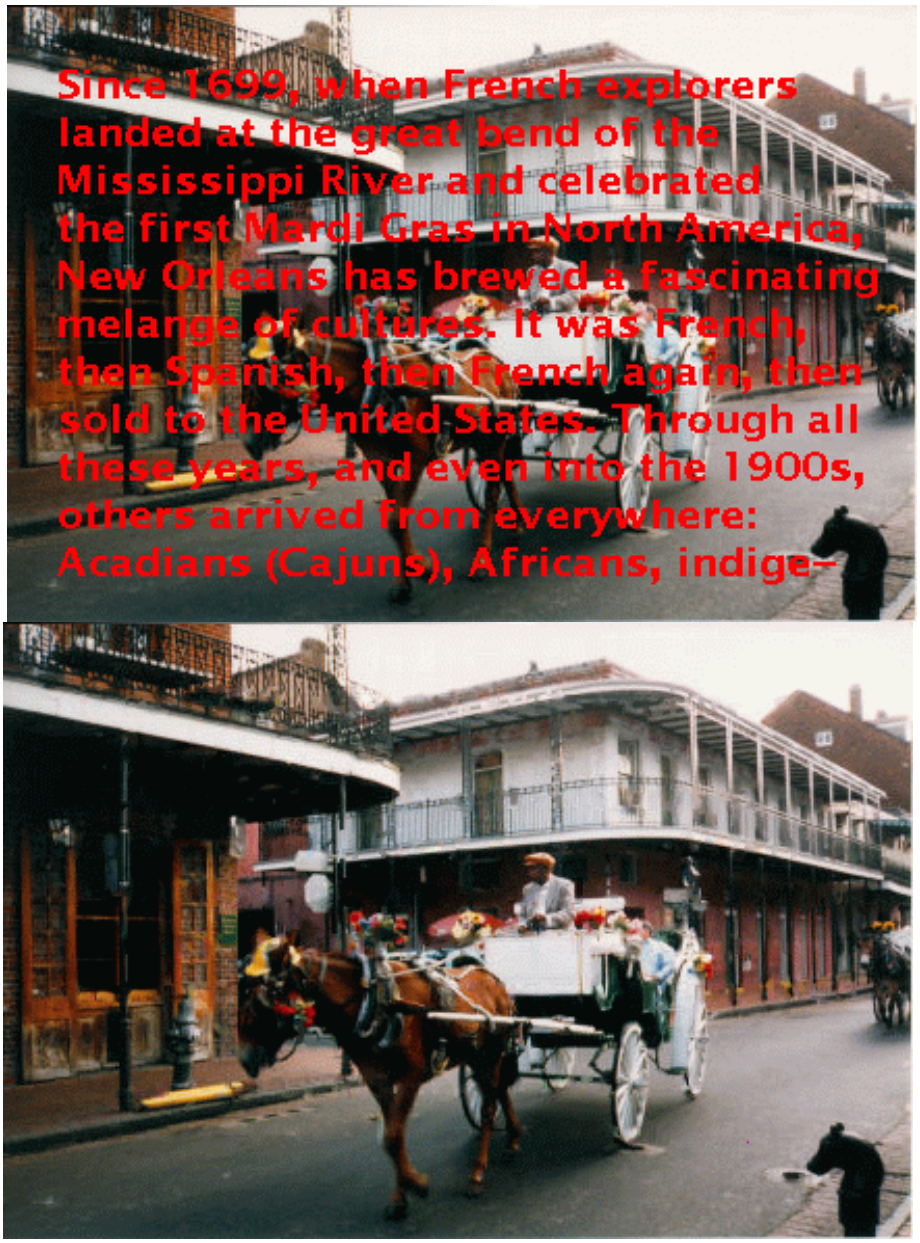

Figure 3.14: Removal of superimposed text.

Figure 3.15: Progressive nature of the algorithm. Several intermediate steps of the inpainting of figure 3.12 are shown.

Figure 3.16: The bungee cord and the knot tying the man's feet have been removed.

Figure 3.17: Limitations of the algorithm: texture is not reproduced.

Chapter 4

Solving Partial Differential Equations on Implicit Surfaces

A novel framework for solving variational problems and partial differential equations for scalar and vector-valued data defined on surfaces is introduced in this chapter. The key idea is to implicitly represent the surface as the level set of a higher dimensional function, and solve the surface equations in a fixed Cartesian coordinate system using this new embedding function. This thereby eliminates the need for performing complicated and not-accurate computations on triangulated surfaces, as it is commonly done in the graphics and numerical analysis literature. We describe the framework and present examples in texture synthesis, flow field visualization, as well as image and vector field regularization for data defined on 3D surfaces.

4.1 Introduction

In a number of computer graphics applications, variational problems and partial differential equations (PDE's) need to be solved for data defined on arbitrary manifolds, three dimensional surfaces in particular. Examples of this are texture synthesis [97, 104], vector field visualization [25], and weathering [26]. In addition, data defined on surfaces often needs to be regularized, e.g., as part of a vector field computation or interpolation process [78, 101], for inverse problems [33], or for surface parameterization [27]. These last examples can be addressed by solving a variational problem on the surface, or its corresponding gradient-descent flow on the surface, using for example the theory of harmonic maps [29], which has recently been demonstrated to be of use for computer graphics applications as well, e.g., [27, 89, 108]. All these equations are generally solved on

triangulated or polygonal surfaces. That is, the surface is given in polygonal (triangulated) form, and the data is discretely defined on it. This involves the non-trivial discretization of the equations in general polygonal grids, as well as the difficult numerical computation of other quantities like projections onto the discretized surface (when computing gradients and Laplacians for example). Although the use of triangulated surfaces is extremely popular in computer graphics, there is still *not* a widely accepted technique to compute differential characteristics such as tangents, normals, principal directions, and curvatures; see for example [24, 64, 91] for a few of the approaches in this direction. On the other hand, it is widely accepted that computing these objects for iso-surfaces (implicit representations) is straightforward and much more accurate and robust. This problem in triangulated surfaces becomes even bigger when we not only have to compute these first and second order differential characteristics of the surface, but also have to use them to solve variational problems and PDE's for data defined on the surface. Moreover, virtually no analysis exists on numerical PDE's on non-uniform grids in the generality needed for computer graphics, making it difficult to understand the behavior of the numerical implementation and its proximity (or lack thereof) to the continuous model.

In this chapter we present a new framework to solve variational problems and PDE's for scalar and vector-valued data defined on surfaces. We use, instead of a triangulated/polygonal representation, an implicit representation: our surface will be the zero-level set of a higher dimensional *embedding* function (i.e., a 3D volume with real values, positive outside the surface and negative inside it). Implicit surfaces have been widely used in computer graphics, e.g., [13, 34, 103], as an alternative representation to triangulated surfaces. We smoothly extend the original (scalar or vector-valued) data lying on the surface to the 3D volume, adapt our PDE's accordingly, and then perform all the computations on the Cartesian grid corresponding to the embedding function. These computations are nevertheless intrinsic to the surface. The advantages of using the Cartesian grid instead of a triangulated mesh are many: we can use well studied numerical techniques, with accurate error measures; the topology of the underlying surface it is not an issue; and we can derive simple, accurate, robust and elegant implementations. If the original surface is not already in implicit form, and it is for example triangulated, we can use any of a number of implicitation algorithms that achieve this representation given a triangulated input, e.g., [28, 50, 90, 106]. For example, the public domain software [60] can be used. If the data is just defined on the surface, an extension of it to the whole volume is also easily achieved using a PDE, as we will see. Therefore, the method here proposed works as well for non-implicit surfaces after the preprocessing is performed. This preprocessing is quite simple and no complicated regridding needs to be done to

go from one surface representation to another (see below). Finally, we will solve the variational problem or PDE only in a band surrounding the zero level set (a classical approach; see [75]). Therefore, although we will be increasing by one the dimension of the space, the computations remain of the same complexity, while the accuracy and simplicity are significantly improved.

4.1.1 Our contribution

Representing *deforming* surfaces as level sets of higher dimensional functions was introduced in [70] as a very efficient technique for numerically studying the deformation (see also [103] for studies on the deformation and manipulation of implicit surfaces for graphics applications). As we have seen in previous chapters, the idea is to represent the surface deformation via the embedding function deformation, which adds accuracy, robustness, and, as expected, topological liberty. When the velocity of the deformation is given by the minimization of an energy, the authors in [109] proposed a "variational level set" method, where they extended the energy (originally defined only on the surface) to the whole space. This allows for the implementation to be in the Cartesian grid. The key of this approach is to go from a "surface energy" to a "volume energy" by using a Dirac's delta function that concentrates the penalization on the given surface.

We will follow this general direction with our fixed, *non deforming* surfaces. In our case, what is being "deformed" is the (scalar or vector-valued) data on the surface. If this deformation is given by an energy-minimization problem (as is the case in data smoothing applications), we will extend the definition of the energy to the whole 3D space, and its minimization will be achieved with a PDE, which despite its being intrinsic to the underlying surface, it is also defined in the whole space. Therefore, it is easily implementable. This is straightforward, as opposed to approaches where one maps the surface data onto the plane, performs the required operations there and then maps the results back onto the triangulated representation of the surface; or approaches that attempt to solve the problem directly on a polygonal surface.

Very interestingly, the new framework proposed here also tells us how to translate into surface terms PDE's that we know that work on the plane but which do not necessarily minimize an energy (e.g., texture synthesis or flow visualization PDE's). Instead of running these PDE's on the plane an then mapping the results onto a triangulated representation of the surface, or running them directly on the triangulated domain, we obtain a 3D straightforward Cartesian grid realization that implements the equation intrinsically on the surface and whose accuracy depends only on the degree of spatial resolution.

Moreover, we consider that for computing differential characteristics and solving PDE's even

for triangulated surfaces, it might be appropriate to run an implicitation algorithm as any of the ones used for the examples in this chapter and then work on the implicit representation. Current algorithms for doing this, some of them publically available [60], are extremely accurate and efficient.

The contribution of this chapter is then a new technique to efficiently solve a common problem in many computer graphics applications: the implementation of PDE's on 3D surfaces. In particular, we show how to transform any intrinsic variational or PDE equation into its corresponding one for implicit surfaces. In this chapter we are then proposing a new framework to better solve existent problems and to help in building up the solutions for new ones. To exemplify the technique and its generality, we implement and extend popular equations previously reported in the literature. Here we solve them with our framework, while in the literature were solved with elaborated discretizations on triangulated representations.

4.2 The general framework

As mentioned before, our approach requires us to have an implicit representation of the given fixed surface, and the data must be defined in a band surrounding it and not just on the surface. The implicit surfaces used in this chapter have been derived from public-domain triangulated surfaces via the computation of a (signed) distance function $\psi(x, y, z)$ to the surface $\mathcal{S}$. Arriving at an implicit representation from a triangulated one is not an issue, there are publicly available algorithms that achieve it in a very efficient fashion. To exemplify this, we have used several of these techniques. For some surfaces the classical Hamilton-Jacobi equation $\| \nabla\psi \| = 1$ was solved on a pre-defined grid enclosing the given surface via the computationally optimal approach devised in [94]. Accurate implicit surfaces from triangulations of the order of one million triangles are obtained in less than two minutes of CPU-time with this technique. Alternatively we used the implementation of the Closest Point Transform available in [60]. The teapot and knot surfaces were obtained from unorganized data points using the technique devised in [110]. We therefore assume from now on that the three dimensional surface $\mathcal{S}$ of interest is given in implicit form, as the zero level set of a given function $\psi : I\!R^3 \rightarrow I\!R$. This function is negative inside the closed bounded region defined by $\mathcal{S}$, positive outside, Lipschitz continuous a.e., with $\mathcal{S} \equiv \{x \in I\!R^3 : \psi(x) = 0\}$. To ensure that the data, which needs not to be defined outside of the surface originally, is now defined in the whole band, one simple possibility is to extend this data u defined on $\mathcal{S}$ (i.e the zero level set of ψ) in such a form that it is constant normal to each level set of ψ. This means the extension

satisfies $\nabla u \cdot \nabla \psi = 0$. (For simplicity, we assume now u to be a scalar function, although we will also address in this chapter problems where the data defined on $\mathcal{S}$ is vector-valued. This is solved in an analogous fashion.) To solve this we numerically search for the steady state solution of the Cartesian PDE

$$\frac{\partial u}{\partial t} + \text{sign}(\psi)(\nabla u \cdot \nabla \psi) = 0.$$

This technique was first proposed and used in [21]. Note that this keeps the given data u on the zero level set of ψ (the given surface) unchanged.

Both the implicitation and data extension (if required at all by the given data), need to be done only once off line. Moreover, they will remain for all applications that need this type of data.

We will exemplify our framework with the simplest case, the heat flow or Laplace equation for scalar data defined on a surface. For scalar data u defined on the plane, that is, $u(x, y) : I\!R^2 \to I\!R$ it is well known that the heat flow

$$\frac{\partial u}{\partial t} = \Delta u \tag{4.1}$$

where $\Delta := \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ is the Laplacian, is the gradient descent flow of the Dirichlet integral

$$\frac{1}{2} \int_{I\!R^2} \| \nabla u \|^2 \, dxdy, \tag{4.2}$$

where ∇ is the gradient.

Eq. (4.1) performs isotropic smoothing of the scalar data u, and this smoothing process progressively decreases the energy defined in eq. (4.2). Figure 4.1 shows a planar scalar image (left), the result of running eq. (4.1) on it for a certain time t_0 (middle), and the result of running eq. (4.1) on it for a certain time $t_1 > t_0$ (right).

If we now want to smooth scalar data u defined on a surface $\mathcal{S}$, we must find the minimizer of the energy given by

$$\frac{1}{2} \int_{\mathcal{S}} \| \nabla_{\mathcal{S}} u \|^2 \, d\mathcal{S}, \tag{4.3}$$

The equation that minimizes this energy is its gradient descent flow:

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u. \tag{4.4}$$

Here $\nabla_{\mathcal{S}}$ is the intrinsic gradient and $\Delta_{\mathcal{S}}$ the intrinsic Laplacian or Laplace-Beltrami operator. These are classical concepts in differential geometry, and basically mean the natural extensions of the gradient and Laplacian respectively, considering all derivatives intrinsic to the surface. For instance, the intrinsic gradient is just the projection onto $\mathcal{S}$ of the regular 3D gradient.

Classically, both in the computer graphics and numerical analysis communities, eq. (4.4) would be implemented in a triangulated surface, giving place to sophisticated and elaborated algorithms even for such simple flows. We now show how to simplify this when considering implicit representations.

Recall that $\mathcal{S}$ is given as the zero level set of a function $\psi : I\!R^3 \to I\!R$, ψ is negative inside the region bounded by $\mathcal{S}$, positive outside with $\mathcal{S} \equiv \{x \in I\!R^3 : \psi(x) = 0\}$. We proceed now to redefine the above energy and compute its corresponding gradient descent flow. Let $\vec{v}$ be a generic three dimensional vector, and $P_{\vec{v}}$ the operator that projects a given three dimensional vector onto the plane orthogonal to $\vec{v}$:

$$P_{\vec{v}} := I - \frac{\vec{v} \otimes \vec{v}}{\| \vec{v} \|^2} \tag{4.5}$$

It is then easy to show that the harmonic energy (4.3) is equivalent to (see for example [86])

$$\frac{1}{2} \int_{\mathcal{S}} \| P_{\vec{N}} \nabla u \|^2 \, d\mathcal{S}, \tag{4.6}$$

where $\vec{N}$ is the normal to the surface $\mathcal{S}$. In other words, $\nabla_{\mathcal{S}} u = P_{\vec{N}} \nabla u$. That is, the gradient intrinsic to the surface ($\nabla_{\mathcal{S}}$) is just the projection onto the surface of the 3D Cartesian (classical) gradient ∇. We now embed this in the function ψ:

$$\begin{aligned} & \tfrac{1}{2} \int_{\mathcal{S}} \| \nabla_{\mathcal{S}} u \|^2 \, d\mathcal{S} = \frac{1}{2} \int_{\mathcal{S}} \| P_{\vec{N}} \nabla u \|^2 \, d\mathcal{S} \\ = & \tfrac{1}{2} \int_{\Omega \in I\!R^3} \| P_{\nabla \psi} \nabla u \|^2 \, \delta(\psi) \| \nabla \psi \| \, dx, \end{aligned}$$

where $\delta(\cdot)$ stands for the delta of Dirac, and all the expressions above are considered in the sense of distributions. Note that first we got rid of intrinsic derivatives by replacing $\nabla_{\mathcal{S}}$ by $P_{\vec{N}} \nabla u$ (or $P_{\nabla \psi} \nabla u$) and then replaced the intrinsic integration ($\int_{\mathcal{S}} d\mathcal{S}$) by the explicit one ($\int_{\Omega \in I\!R^3} dx$) using the delta function. Intuitively, although the energy lives in the full space, the delta function forces the penalty to be effective only on the level set of interest. The last equality includes the embedding, and it is based on the following simple facts:

1. $\nabla \psi \parallel \vec{N}$.
2. $\int_\Omega \delta(\psi) \| \nabla \psi \| \, dx = \int_{\mathcal{S}} d\mathcal{S} =$ surface area.

In Appendix B we show that the gradient descent of this energy is given by

$$\frac{\partial u}{\partial t} = \frac{1}{\| \nabla \psi \|} \nabla \cdot (P_{\nabla \psi} \nabla u \parallel \nabla \psi \parallel). \tag{4.7}$$

In other words, this equation corresponds to the intrinsic heat flow for data on an implicit surface. But all the gradients in this PDE are defined in the three dimensional Cartesian space, not in the surface $\mathcal{S}$ (this is why we need the data to be defined at least on a band around the surface). The numerical implementation is then straightforward. This is the beauty of the approach! Basically, for this equation we use a classical scheme of forward differences in time and a succession of forward and backward differences in space (see Appendix E for details). The other equations in this chapter are similarly implemented. This follows techniques as those in [82]. Once again, due to the implicit representation, classic numerics are used, avoiding elaborate projections onto discrete surfaces and discretization on general meshes, e.g., [24, 40].

It is easy to show a number of important properties of this equation:

1. For any second embedding function $\phi = \phi(\psi)$, with $\phi' \neq 0$, we obtain the same gradient descent flow. Since both ψ and ϕ have to share the zero level set, and we are only interested in the flow around this zero level set, this means that the flow is (locally) independent of the embedding function.[1]

2. If ψ is the signed distance function, a very popular implicit representation of surfaces (obtained for example from the implicitation algorithms previously mentioned), the gradient descent simplifies to
$$\frac{\partial u}{\partial t} = \nabla \cdot (P_{\nabla\psi} \nabla u). \tag{4.8}$$

We note that we could also have derived eq. (4.7) directly from the harmonic maps flow

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u,$$

via the simple geometry exercise of computing $\Delta_{\mathcal{S}} u$ for $\mathcal{S}$ in implicit form. This last property is of particular significance. It basically shows how to solve general PDE's, not necessarily gradient-descent flows, for data defined on implicit surfaces. All that we need to do is to recompute the components of the PDE for implicit representations of the surface. Note that in this way, conceptually, we can re-define classical planar PDE's on implicit surfaces, making them both intrinsic to the underlying surface and defined on the whole space.

From this very simple example we have seen the key point of our approach. If the process that we want to implement comes from the minimization of an energy, we derive a PDE for the whole space by computing the gradient-descent of the whole-space-extension of that energy. Otherwise,

[1] We thank F. Mémoli for helping with this fact.

given a planar PDE we recompute its components for an implicit representation of the surface. For instance, anisotropic diffusion can be performed on the plane by

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(\frac{\nabla u}{\parallel \nabla u \parallel} \right), \tag{4.9}$$

which minimizes the energy $\frac{1}{2} \int_{I\!R^2} \parallel \nabla u \parallel \, dxdy$ (see [82]).

Figure 4.2 shows a planar scalar image (left), the result of running eq. (4.9) on it for a certain time t_0 (middle), and the result of running eq. (4.9) on it for a certain time $t_1 > t_0$ (right).

If we now want to perform anisotropic diffusion of scalar data on a surface $\mathcal{S}$, we can either recompute the gradient-descent flow for an extension of the energy or just substitute in eq. (4.9) the corresponding expressions. Either way, we obtain the same result, the following PDE, which is valid in the Euclidean space:

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla \psi \parallel} \nabla \cdot \left(\frac{P_{\nabla \psi} \nabla u}{\parallel P_{\nabla \psi} \nabla u \parallel} \parallel \nabla \psi \parallel \right). \tag{4.10}$$

In the following section more equations will be presented.

4.3 Experimental examples

We now exemplify the framework just introduced for a number of important cases. The numerical implementation used is quite simple, and requires a few lines of C++ code. The CPU time required for the diffusion examples is of a few seconds on a PC (512Mb RAM, 1GHz) under Linux. For the texture synthesis examples, the CPU time ranges from a few minutes to one hour, depending on the pattern and parameters chosen. All the volumes used contain roughly 128^3 voxels. Note once again that due to the use of only a narrow band surrounding the zero level set, the order of the algorithmic complexity remains the same. On the other hand, the use of straightforward Cartesian numerics reduces the overall algorithmic complexity, improving accuracy and simplifying the implementation.

4.3.1 Diffusion of scalar images on surfaces

The use of PDE's for image enhancement has become one of the most active research areas in image processing [17]. In particular, diffusion equations are commonly used for image regularization, denoising, and multiscale representations (representing the image simultaneously at several scales

or levels of resolution). This started with the works in [51, 102], where the authors suggested the use of the linear heat flow (4.1) for this task, where u represents the image gray values (the original image is used as initial condition). As we have seen, this is the gradient-descent of (4.2), and the generalizations of these equations for data on the surface are given by (4.4) and (4.3) respectively. In implicit form, the heat flow on surfaces is given by (4.7). Figure 4.3 shows a simple example of image diffusion on a surface. Please note that this is *not* equivalent to performing 3D smoothing of the data and then looking to see what happened on $\mathcal{S}$. Our flow, though using extended 3D data, performs smoothing *directly* on the surface, it is an intrinsic heat flow. The complete details of the numerical implementation of this flow are given in Appendix E (this will once again show how the implementation is significantly simplified with the framework here described).

Figure 4.4 shows an example of adding a constraint to the surface PDE, following [82]. In this case, the variance of the noise is known and this is added to the variational formulation. To the flow (4.10) we add

$$\lambda(\psi - \psi_0),$$

which comes from the Euler-Lagrange when the constraint $\frac{\lambda}{2}\int_{\mathcal{S}}(u-u_0)^2 d\mathcal{S}$ (or $\frac{\lambda}{2}\int_{I\!R^3}(u-u_0)^2\delta(\psi) \parallel \nabla\psi \parallel dx$) is added to the harmonic energy (λ is a parameter and u_0 is the initial noisy image; see Appendix C for a way to estimate λ). In the same figure, compare the results obtained when no constraint is imposed.

The same approach, that of anisotropic diffusion with a stopping term, may be used to perform intrinsic *deblurring*, see [22].

We should note before proceeding that [48] also showed how to regularize images defined on a surface. The author's approach is limited to graphs (not generic surfaces) and only applies to level set based motions. The approach is simply to project the deformation of the data on the surface onto a deformation on the plane.

4.3.2 Diffusion of directional data on surfaces

A particularly interesting example is obtained when we have unit vectors defined on the surface. That is, we have data of the form $u : \mathcal{S} \to S^{n-1}$. When $n = 3$ our unit vectors lie on the sphere. Particular examples of this are principal directions (or general directional fields on 3D surfaces) and chromaticity vectors (normalized RGB vectors). This is also one of the most studied cases of the theory of harmonic maps due to its physical relationship with liquid crystals, and it was introduced in [89] for the regularization of directional data, unit vectors, on the plane. This framework of

harmonic maps was used in computer graphics for texture mapping and surface parameterization, as pointed out earlier.

We still want to minimize an energy of the form

$$\int_{\mathcal{S}} \parallel \nabla_{\mathcal{S}} u \parallel^p d\mathcal{S},$$

though in this case $\nabla_{\mathcal{S}}$ is the vectorial gradient and the minimizer is restricted to be a unit vector. It is easy to show, e.g., [15, 88], that the gradient descent of this energy is given by the coupled system of PDE's

$$\frac{\partial u_i}{\partial t} = \text{div}_{\mathcal{S}} \left(\parallel \nabla_{\mathcal{S}} u \parallel^{p-2} \nabla_{\mathcal{S}} u_i \right) + u_i \parallel \nabla_{\mathcal{S}} u \parallel^p, \;\; 1 \leq i \leq n.$$

This flow guarantees that the initial unit vector $u(x, y, z, 0)$ remains a unit vector $u(x, y, z, t)$ all the time, thereby providing an equation for isotropic ($p = 2$) and anisotropic ($p = 1$) diffusion and regularization of unit vectors on a surface.

We can now proceed as before, and embed the surface $\mathcal{S}$ into the zero level-set of ψ, obtaining the following gradient descent flows:

$$\frac{\partial u_i}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot \left(\frac{P_{\nabla\psi} \nabla u_i}{\parallel P_{\nabla\psi} \nabla u \parallel^{2-p}} \parallel \nabla\psi \parallel \right) + u_i \parallel P_{\nabla\psi} \nabla u \parallel^p . \tag{4.11}$$

See Appendix D for a derivation of this result for $p = 2$. Note once again that although the regularization is done intrinsically on the surface, this equation only contains Cartesian gradients. An example of this flow for anisotropic diffusion of principal direction vectors is given in Figure 4.5. On the left, we see the surface of a bunny with its correspondent vector field for the major principal direction. Any irregularity on the surface produces a noticeable alteration of this field, as can be seen in the details *a* and *b*. In the details *a'* and *b'*, we see the result of applying the flow (4.11). Once again, the implementation of this flow with our framework is straightforward, while it would require very sophisticated techniques on triangulated surfaces (techniques that, in addition, are not supported by theoretical results).

Following also the work [89] for color images defined on the plane, we show in Figure 4.6 how to denoise a color image painted on an implicit surface. The basic idea is to normalize the RGB vector (a three dimensional vector) to a unit vector representing the chroma, and diffuse this unit vector with the harmonic maps flow (4.11).[2] The corresponding magnitude, representing the

[2]We re-normalize at every discrete step of the numerical evolution to address deviations from the unit norm due to numerical errors [23]. We could also extend the framework in [1] and apply it to our equations.

brightness, is smoothed separately via scalar diffusion flows as those presented before (e.g., the intrinsic heat flow or the intrinsic anisotropic heat flow). That is, we have to regularize a map onto S^2 (the chroma) and another one onto $\mathbb{R}$ (the brightness).

4.3.3 Pattern formation on surfaces via reaction-diffusion flows

The use of reaction-diffusion equations for texture synthesis became very popular in computer graphics following the works of Turk [97] and Witkin and Kass [104]. These works follow original ideas by Turing [96], who showed how reaction diffusion equations can be used to generate patterns. The basic idea in these models is to have a number of "chemicals" that diffuse at different rates and that react with each other. The pattern is then synthesized by assigning a brightness value to the concentration of one of the chemicals. The authors in [97, 104] used their equations for planar textures and textures on triangulated surfaces. By using the framework here described, we can simply create textures on (implicit/implicitized) surfaces, without the elaborated schemes developed in those papers.[3]

Assuming a simple isotropic model with just two chemicals u_1 and u_2, we have

$$\frac{\partial u_1}{\partial t} = F(u_1, u_2) + D_1 \Delta u_1,$$

$$\frac{\partial u_2}{\partial t} = G(u_1, u_2) + D_2 \Delta u_1,$$

where D_1 and D_2 are two constants representing the diffusion rates and F and G are the functions that model the reaction.

Introducing our framework, if u_1 and u_2 are defined on a surface $\mathcal{S}$ implicitly represented as the zero level set of ψ we have

$$\frac{\partial u_1}{\partial t} = F(u_1, u_2) + D_1 \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot (P_{\nabla\psi} \nabla u_1 \parallel \nabla\psi \parallel), \tag{4.12}$$

$$\frac{\partial u_2}{\partial t} = G(u_1, u_2) + D_2 \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot (P_{\nabla\psi} \nabla u_2 \parallel \nabla\psi \parallel). \tag{4.13}$$

For simple isotropic patterns, Turk [97] selected

$$F(u_1, u_2) = s(16 - u_1 u_2),$$

[3]Note that this is not the scheme proposed in [76], where the texture is created in the full 3D space. Here, the texture is created via reaction-diffusion flows intrinsic to the surface, just the implementation is on the embedding 3D space.

$$G(u_1, u_2) = s(u_1 u_2 - u_2 - \beta),$$

where s is a constant and β is a random function representing irregularities in the chemical concentration. Examples of this, for implicit surfaces, are given in Figure 4.7 (the coupled PDE's shown above are run until steady state is achieved). To simulate anisotropic textures, instead of using additional chemicals as in [97], we use anisotropic diffusion, as suggested in [104]. For this purpose, we replace eq. (4.12) with:

$$\frac{\partial u_1}{\partial t} = F(u_1, u_2) + D_1 \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot ((\overrightarrow{d} \cdot P_{\nabla\psi} \nabla u_1) \overrightarrow{d} \parallel \nabla\psi \parallel), \tag{4.14}$$

where $\overrightarrow{d}$ is a vector field tangent to the surface, e.g., the field of the major principal direction (which for our examples has been also accurately computed directly on the implicit surface, using the technique proposed in [63]). Note how this particular selection of the anisotropic reaction-diffusion flow direction provides a texture that helps on the shape perception of the object. Additional patterns can be obtained with different combinations of the reaction and diffusion parts of the flow.

4.3.4 Flow visualization on 3D surfaces

Inspired by the work on line integral convolution [16] and that on anisotropic diffusion [77], the authors of [25] suggested to use anisotropic diffusion to visualize flows in 2D and 3D. The basic idea is, starting from a random image, anisotropically diffuse it in the directions dictated by the flow field. The authors presented very nice results both in 2D (flows on the plane) and 3D (flows on a surface), but once again using triangulated surfaces which introduce many computational difficulties. In a straightforward fashion we can compute these anisotropic diffusion equations on the implicit surfaces with the framework here introduced. The equation used is simply

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot ((\overrightarrow{d} \cdot P_{\nabla\psi} \nabla u) \overrightarrow{d} \parallel \nabla\psi \parallel), \tag{4.15}$$

Some results are presented in Figure 4.8. Note the complicated topology and how both the inside and outside parts of the surfaces are easily handled with our implicit approach. Also note that, when we choose the vector field to be that of one of the principal directions, the result emphasizes the surface shape.

Figure 4.1: Planar isotropic diffusion. Left: original image. Middle and Right: increasing number of diffusion steps.

Figure 4.2: Planar anisotropic diffusion. Left: original image. Middle and Right: increasing number of anisotropic diffusion steps.

Figure 4.3: Intrinsic isotropic diffusion. Left: original image. Middle: after 60 diffusion steps. Right: after 160 diffusion steps.

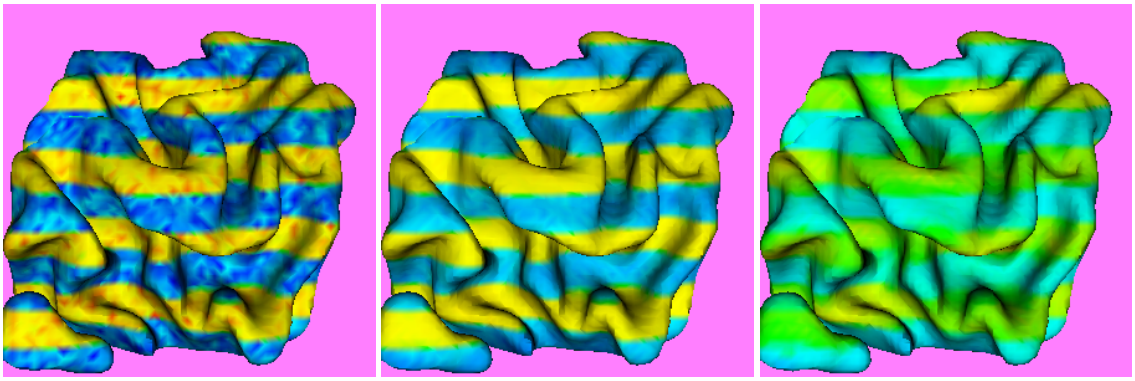

Figure 4.4: Intrinsic Total Variation (TV) denoising (anisotropic diffusion with stopping term). Scalar data shown in color for visualization purposes. Left: original. Middle:TV at step 80. Right: intrinsic anisotropic diffusion, with no stopping term, at step 80. Notice how TV does not smear the data.

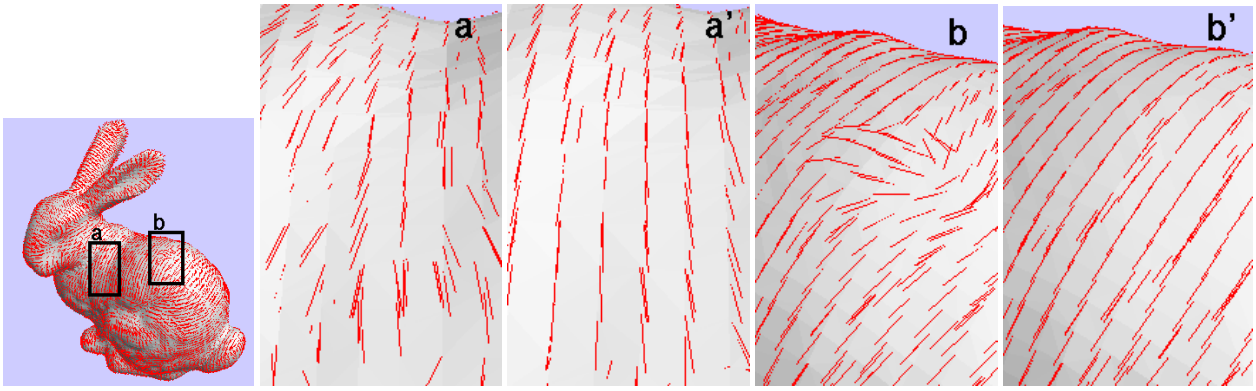

Figure 4.5: Intrinsic vector field regularization. Left: original field of major principal direction of the surface. Details *a* and *b*: original field. Details *a'* and *b'*: after anisotropic regularization.

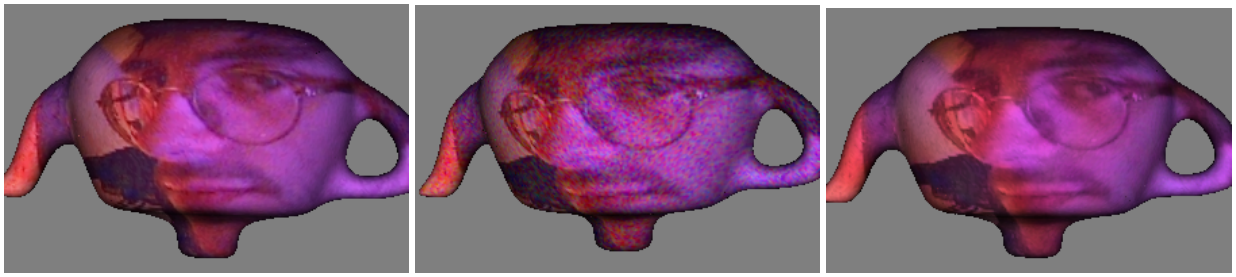

Figure 4.6: Intrinsic vector field regularization. Left: original color image. Middle: heavy noise has been added to the 3 color channels. Right: color image reconstructed after 20 steps of anisotropic diffusion of the chroma vectors.

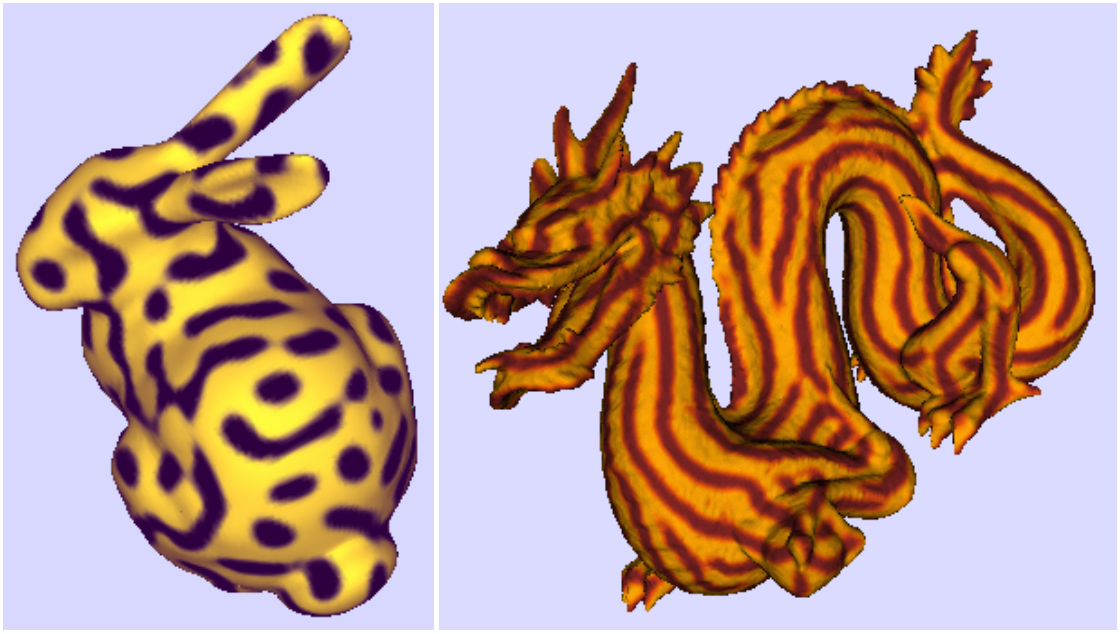

Figure 4.7: Texture synthesis via intrinsic reaction-diffusion flows on implicit surfaces. Left: isotropic. Right: anisotropic. Pseudo-color representation of scalar data is used. The numerical values used in the computations were $D_1 = 1.0$, $D_2 = 0.0625$, $s = 0.025$, $\beta = 12.0 \pm 0.1$, $u_1(0) = u_2(0) = 4.0$.

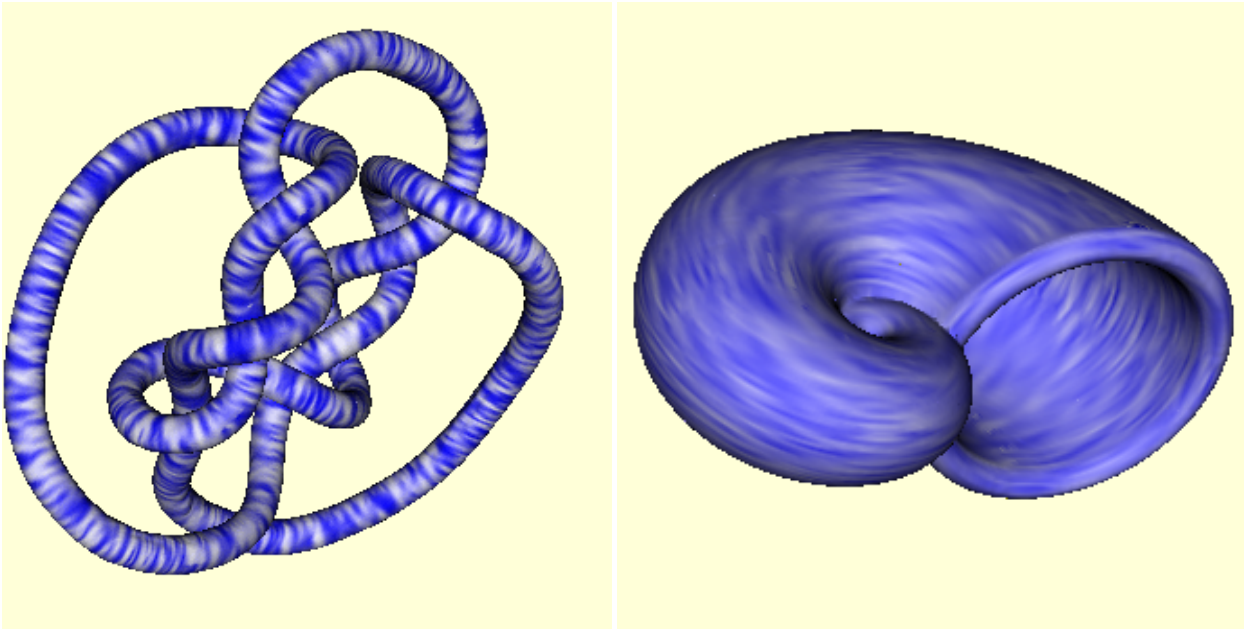

Figure 4.8: Flow visualization on implicit 3D surfaces via intrinsic anisotropic diffusion flows. Left: flow aligned with the major principal direction of the surface. Right: flow aligned with the minor principal direction of the surface. Pseudo-color representation of scalar data is used.

Chapter 5

Conclusion and future research

In this chapter we will briefly summarize the concepts previously introduced. Also, future lines of research are suggested, both to improve performance and to extend the present techniques to new applications.

5.1 Preliminary work

In Chapter 2 we presented two algorithms that use a system of coupled PDE's and projection of velocities to perform tracking. This concept of projecting quantities was later used both for the inpainting equation (Chapter 3) and for the framework for solving PDE's on implicit surfaces (Chapter 4). The key idea is that the projection of velocities makes our *tracking* surface have the same velocity as that of the object to be tracked. The formulation is very general and allows for improvement, specially in the Morphing Active Contours algorithm: finding more robust selections of the feature map $\mathcal{F}_i$ and the discrepancy function β, using more than just 2 frames (via Kalman filtering or using the techniques in the novel scheme developed in [12]), proving existence and uniqueness theoretical results.

5.2 Image Inpainting

In Chapter 3 we introduced a novel algorithm for image inpainting that attempts to replicate the basic techniques used by professional restorators. The basic idea is to smoothly propagate information from the surrounding areas in the isophotes direction. The user needs only to provide the

region to be inpainted, the rest is automatically performed by the algorithm in a few minutes. The inpainted images are sharp and without color artifacts. The examples shown suggest a wide range of applications like restoration of old photographs and damaged film, removal of superimposed text, and removal of objects. The results can either be adopted as a final restoration or be used to provide an initial point for manual restoration, thereby reducing the total restoration time by orders of magnitude.

One of the main problems with our technique is the reproduction of large textured regions, as can be seen in Figure 3.17. The algorithm here proposed is currently being tested in conjunction with texture synthesis ideas to address this issue. We are mainly investigating the combination of this approach with the reaction-diffusion ideas of Kass and Witkin and of Turk, and the texture synthesis ideas of Efros [30]. An ideal algorithm should be able to automatically switch between textured and geometric areas, and select the best suited technique for each region. This is currently being investigated, for the specific application of recovery of losses in JPEG compression, by S. Rane (Graduate Student at the ECE Department.)

We would also like to investigate how to inpaint from partial degradation. In the example of the old photo for example, ideally the mask should not be binary, since some underlying information exists in the degraded areas.

Although theoretical results for high order equations are available, e.g., [10], and some properties like preservation of the image moments can be immediately proved for our corresponding equation (this was done by A. Bertozzi), further formal study of our inpainting equation is needed (see also [95, 19]). Nevertheless, this suggests the investigation of the use of lower, second order, PDE's to address the inpainting problem. We can split the inpainting problem into two coupled variational formulations, one for the isophotes direction (point (2) in Section 3.3.1) and one for the gray-values, consistent with the estimated directions (point (3) in Section 3.3.1). The corresponding gradient descent flows will give two coupled second order PDE's for which formal results regarding existence and uniqueness of the solutions can be shown. An alternative formulation joins both equations in a single PDE. This is reported in [5].

Finally, let us comment on the noted link between the inpainting equation and the Navier-Stokes one. Finding a correspondence in terminology for Image Processing and Fluid Dynamics may prove to be of significance, helping to understand some problems by looking at them in a different way. We can use tools that have already been developed with great success: in our case, we will try state of the art numerical implementations of the Navier-Stokes equation for our inpainting problem, hopefully increasing speed and performance. The fact that the Navier-Stokes

equation allows for spatial periodicity in its solutions may be a way to incorporate texture in our problem. This analogy with the physical world might also help finding a way to extend inpainting for new applications: inpainting in video, high quality digital zooming, image/video compression.

5.3 PDE's on implicit surfaces

In this chapter we introduced a novel framework for solving variational problems and PDE's for data defined on surfaces. The technique borrows ideas from the level set theory and the theory of harmonic maps. The surface is embedded in a higher dimensional function, and the Euler-Lagrange flow or PDE is solved in the Cartesian coordinate system of this embedding function. The equations are intrinsic to the implicit surface, following the general formulations in harmonic map theory. With this framework we enjoy accuracy, robustness, and simplicity, as expected from the computation of differential characteristics on iso-surfaces (implicit surfaces). In addition to presenting the general approach, we have exemplified it with equations arising in image processing and computer graphics. We are currently investigating other applications of this framework, e.g, image inpainting on surfaces [9], inverse problems as those in [33], and texture mapping for implicit surfaces following [27].

Appendix A

Numerical implementation of the Inpainting Equation

```
/*****      main ***********************/
fr: image
B: mask (=1 inside inpainting region, 0 outside)
dt: time step
T: total # of iterations
pasosK= # of anisotropic difusion steps
pasosL= # of inpainting steps in between difusion steps

for(int t=0; t<T;t++)
 if(div(t,pasosL).rem!=0)
  {
 ut_tmp2=fr->Laplaciano(); //ut_tmp2 is the Laplacian of fr
 ut_tmp=fr->inpaint(B,ut_tmp2);
 ut=ut_tmp*dt;
 fr=fr+ut;
 delete ut_tmp2;
 delete ut_tmp;
   }
    else
     fr->difusion(pasosK,B);
```

```
/********** laplacian **************/

f: original image
LAP: laplacian of f

  for (int i=1;   i < rows  ;i++)
      for (int j=1;j < cols; j++)
{
  fxx=f[i+1][j]-2*f[i][j]+f[i-1][j];
  fyy=f[i][j+1]-2*f[i][j]+f[i][j-1];
  LAP[i][j]=fxx+fyy;
}


/************* inpainting **************/


res=im->inpaint(B,lap)
im: original image
B: mask
lap: laplacian of im
res: final result


  for (int j=1;j<cols-1; j++)
    for (int i=1;i <rows-1  ;i++)
      if(B[i][j]) //if we are inside the inpainting region
{
  Ix=( im[i+1][j]-im[i-1][j] )/2;
  Iy=( im[i][j+1]-im[i][j-1] )/2;
  ModGI=sqrt(Ix*Ix+Iy*Iy+1e-6); //ModGI is the norm
//of the gradient of im
  NIx=Ix/ModGI;
```

```
  NIy=Iy/ModGI; //NIx and NIy are the components
//of the normalized gradient

  Ixat=im[i][j]-im[i-1][j];//backward derivative
  Ixad=im[i+1][j]-im[i][j];//forward derivative
  Iyat=im[i][j]-im[i][j-1];
  Iyad=im[i][j+1]-im[i][j];


  Ixatm=min(Ixat,0);
  IxatM=max(Ixat,0);
  Ixadm=min(Ixad,0);
  IxadM=max(Ixad,0);
  Iyatm=min(Iyat,0);
  IyatM=max(Iyat,0);
  Iyadm=min(Iyad,0);
  IyadM=max(Iyad,0);


  c=0.5*( (lap[i+1][j]-lap[i-1][j])*(-NIy) +
(lap[i][j+1]-lap[i][j-1])*(NIx) );

  if (c>0)
    ModGI=sqrt( Ixatm*Ixatm + IxadM*IxadM +
Iyatm*Iyatm + IyadM*IyadM );
  else
    ModGI=sqrt( IxatM*IxatM + Ixadm*Ixadm +
IyatM*IyatM + Iyadm*Iyadm );

  c*=ModGI;

  float ssigno=signo(c);//ssigno is the sign (1 or -1) of c

  res[i][j]=ssigno*sqrt(sqrt(ssigno*c)); //non-linear scaling
  //of the inpainting equation
```

```
/********** anisotropic difusion  ********************/

res=f->difusion(T,B)

f: original image
T: # of steps of difusion
B: mask
res: final result


res=fr;
 for(t=0;t<T;t++)
  for (int j=1;j <cols-1;j++)
    for (int i=1; i<rows-1;i++)
      if(B[i][j])
{
  fx=( f[i+1][j]-f[i-1][j] )/2;
  fy=( f[i][j+1]-f[i][j-1] )/2;
  fxx=( f[i+1][j]-2*f[i][j]+f[i-1][j] );
  fyy=( f[i][j+1]-2*f[i][j]+f[i][j-1] );
  fxy=( f[i+1][j+1]-f[i-1][j+1]-f[i+1][j-1]+f[i-1][j-1] )/4;
  n2=fx*fx+fy*fy+1e-10;
  res[i][j]= f[i][j]+0.2*(fyy*fx*fx+fxx*fy*fy-2*fx*fy*fxy)/n2;
}
```

Appendix B

Heat flow on implicit surfaces

Considering

$$E(u) := \frac{1}{2}\int_{\Omega\in I\!R^3} \parallel P_{\nabla\psi}\nabla u \parallel^2 \delta(\psi) \parallel \nabla\psi \parallel dx,$$

and μ a perturbation of u,

$$\begin{aligned}
\tfrac{d}{dt} \quad |_{t=0}E(u+t\mu) &= \int_\Omega (P_{\nabla\psi}\nabla u \cdot P_{\nabla\psi}\nabla\mu)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&= \int_\Omega \left(P_{\nabla\psi}\nabla u \cdot \left(\nabla\mu - \frac{\nabla\psi\cdot\nabla\mu}{\parallel \nabla\psi \parallel^2}\nabla\psi\right)\right)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&= \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\mu)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&\quad - \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\psi)\frac{\nabla\psi\cdot\nabla\mu}{\parallel \nabla\psi \parallel^2}\delta(\psi) \parallel \nabla\psi \parallel dx \\
&= \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\mu)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&= -\int_\Omega \nabla\cdot(P_{\nabla\psi}\nabla u\delta(\psi) \parallel \nabla\psi \parallel)\mu dx \\
&= -\int_\Omega \nabla\cdot(P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel)\delta(\psi)\mu dx \\
&\quad - \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\psi)\delta'(\psi) \parallel \nabla\psi \parallel \mu dx \\
&= -\int_\Omega \nabla\cdot(P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel)\delta(\psi)\mu dx \\
&= -\int_{\mathcal{S}\equiv\{\psi=0\}} \frac{1}{\parallel \nabla\psi \parallel}\nabla\cdot(P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel)\mu d\mathcal{S}.
\end{aligned}$$

Since the above has to be zero for all μ, we conclude that at the zero level set of ψ,

$$\frac{1}{\parallel \nabla\psi \parallel}\nabla\cdot(P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel) = 0,$$

and we make a natural extension to the whole domain Ω by considering this to hold on it.[1] We then obtain that the gradient descent for the "implicit harmonic energy" is given by

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel}\nabla \cdot (P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel). \tag{B.1}$$

[1]We have assumed that $\parallel \nabla\psi \parallel \neq 0$, at least on a band surrounding the zero level set. This assumption is valid since we can make the embedding function to be a distance function ($\parallel \nabla\psi \parallel = 1$), or simply multiply ψ by another function that guarantees that the zero-level set $\mathcal{S}$ is preserved and that the gradient of the new embedding function is not zero.

Appendix C

Denoising with stopping term

We have added an extra term to equation (4.10) so that the evolution stops by itself, given an estimate of the amount of noise that the original image has. The resulting PDE is:

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot \left(\frac{P_{\nabla\psi}\nabla u}{\parallel P_{\nabla\psi}\nabla u \parallel} \parallel \nabla\psi \parallel -\lambda(u - u_0) \right). \tag{C.1}$$

Assuming Gaussian noise with known variance σ^2 on the surface, $\sigma^2 = \int_{\Omega \in I\!R^3} (u - u_0)^2 \delta(\psi) \parallel \nabla\psi \parallel dx$, we need then to estimate the parameter λ. A way to do this, as suggested in [82], is the following. We merely multiply eq. (C.1) by $(u - u_0)\delta(\psi) \parallel \nabla\psi \parallel$ and integrate by parts over Ω. If the steady state has been reached, the left side of eq. (C.1) vanishes, and we have:

$$\lambda = -\frac{1}{2\sigma^2} \int_{\mathcal{S}} (\nabla u - \nabla u_0) \cdot \frac{P_{\nabla\psi}\nabla u}{\parallel P_{\nabla\psi}\nabla u \parallel} d\mathcal{S} \tag{C.2}$$

This gives us a dynamic value $\lambda(t)$.

Appendix D

Diffusion of directional data on surfaces

We consider u to be data of the form $u : \mathcal{S} \to S^{n-1}$. For $n = 3$ we have unit vectors on the sphere: $u = (u_1, u_2, u_3), \parallel u \parallel = 1$.

We define the norm of the vectorial gradient of u:
$\parallel \nabla u \parallel = (\parallel \nabla u_1 \parallel^2 + \parallel \nabla u_2 \parallel^2 + \parallel \nabla u_3 \parallel^2)^{\frac{1}{2}}$
and also the norm of the intrinsic vectorial gradient:
$\parallel P_{\nabla\psi} \nabla u \parallel = (\parallel P_{\nabla\psi} \nabla u_1 \parallel^2 + \parallel P_{\nabla\psi} \nabla u_2 \parallel^2 + \parallel P_{\nabla\psi} \nabla u_3 \parallel^2)^{\frac{1}{2}}$

We want to minimize the energy

$$E_A(u) := \frac{1}{2} \int_{\Omega \in I\!R^3} \parallel P_{\nabla\psi} \nabla u \parallel^2 \delta(\psi) \parallel \nabla\psi \parallel dx,$$

with the constraint that u is of unit norm:

$$E_B(u) := \frac{1}{2} \gamma \int_{\Omega \in I\!R^3} (u^2 - 1) \delta(\psi) \parallel \nabla\psi \parallel dx,$$

so in practice we want to minimize the energy $E(u) = E_A(u) + E_B(u)$.

Applying to $E_A(u)$ the method described in Appendix B,

$$\begin{aligned} \tfrac{d}{dt} \quad |_{t=0} E(u + t\mu) &= \int_\Omega (P_{\nabla\psi} \nabla u_1 \cdot P_{\nabla\psi} \nabla \mu_1) \delta(\psi) \parallel \nabla\psi \parallel dx \\ &+ \int_\Omega (P_{\nabla\psi} \nabla u_2 \cdot P_{\nabla\psi} \nabla \mu_2) \delta(\psi) \parallel \nabla\psi \parallel dx \\ &+ \int_\Omega (P_{\nabla\psi} \nabla u_3 \cdot P_{\nabla\psi} \nabla \mu_3) \delta(\psi) \parallel \nabla\psi \parallel dx \end{aligned}$$

we obtain that the gradient descent for E_A is given by

$$\frac{\partial u_i}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel}\nabla \cdot (P_{\nabla\psi}\nabla u_i \parallel \nabla\psi \parallel). \tag{D.1}$$

for each of the three components of u.

The gradient descent for E_B is simply

$$\frac{\partial u_i}{\partial t} = -\gamma u. \tag{D.2}$$

So the composed gradient descent, for E, is

$$\frac{\partial u_i}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel}\nabla \cdot (P_{\nabla\psi}\nabla u_i \parallel \nabla\psi \parallel) - \gamma u_i. \tag{D.3}$$

We must find the value of γ. Multiplying both sides of eq. (D.3) by u_i, making a summation over i and bearing in mind that $\parallel u \parallel = 1$, we get

$$\gamma = \frac{1}{\parallel \nabla\psi \parallel}\Sigma u_i \nabla \cdot (\parallel \nabla\psi \parallel P_{\nabla\psi}\nabla u_i) \tag{D.4}$$

Using the equalities

$$\begin{aligned} u_i\nabla \cdot (\parallel \nabla\psi \parallel P_{\nabla\psi}\nabla u_i) &= \nabla \cdot (u_i \parallel \nabla\psi \parallel P_{\nabla\psi}\nabla u_i) - \nabla u_i \cdot \parallel \nabla\psi \parallel P_{\nabla\psi}\nabla u_i, \\ u_i P_{\nabla\psi}\nabla u_i = u_i(\nabla u_i - \frac{\nabla\psi\nabla u_i}{\parallel \nabla\psi \parallel^2}\nabla\psi) &= \frac{1}{2}(\nabla u_i^2 - \frac{\nabla\psi\nabla u_i^2}{\parallel \nabla\psi \parallel^2}\nabla\psi) \\ \nabla u_i \cdot \parallel \nabla\psi \parallel P_{\nabla\psi}\nabla u_i = \parallel \nabla\psi \parallel \nabla u_i \cdot P_{\nabla\psi}\nabla u_i &= \parallel \nabla\psi \parallel \parallel P_{\nabla\psi}\nabla u_i \parallel^2 \end{aligned}$$

and the fact that

$$\Sigma\nabla u_i^2 = \nabla(\Sigma u_i^2) = \nabla(1) = 0, \tag{D.5}$$

we finally obtain

$$\gamma = - \parallel P_{\nabla\psi}\nabla u \parallel^2 \tag{D.6}$$

so the gradient descent for E is

$$\frac{\partial u_i}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel}\nabla \cdot (P_{\nabla\psi}\nabla u_i \parallel \nabla\psi \parallel) + u_i \parallel P_{\nabla\psi}\nabla u \parallel^2 . \tag{D.7}$$

Appendix E

Numerical implementation of the heat flow on implicit surfaces

We now provide details on the numerical implementation of the intrinsic heat flow on implicit surfaces. All other equations reported in this chapter are similarly implemented. Recall that all equations are on Cartesian grids, thereby permitting the use of classical numerics. We work on a cubic grid ($\Delta x = \Delta y = \Delta z = 1$), using an explicit scheme, where we compute the value of $u^n_{i,j,k} = u(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$ based only in previous values ($t = (n-1)\Delta t$) of its neighbors, i.e., forward time differences.

Firstly, we compute the 3D gradient of u using forward differences:

$\overrightarrow{v}^n_{i,j,k} = \nabla_+ u^n_{i,j,k} = (u^n_{i+1,j,k} - u^n_{i,j,k}, u^n_{i,j+1,k} - u^n_{i,j,k}, u^n_{i,j,k+1} - u^n_{i,j,k})$.

We compute the vector $\overrightarrow{N}(i,j,k)$, which gives the direction of the (outward) normal to the isosurface of ψ at the point (i,j,k):

$\overrightarrow{N}_{i,j,k} = \nabla\psi_{i,j,k} = \frac{1}{2}(\psi_{i+1,j,k} - \psi_{i-1,j,k}, \psi_{i,j+1,k} - \psi_{i,j-1,k}, \psi_{i,j,k+1} - \psi_{i,j,k-1})$.

Here we have used central differences. Note that, since ψ is fixed, $\overrightarrow{N}(i,j,k)$ does not change in time and we need only to compute it once. Its norm is: $\| \overrightarrow{N}_{i,j,k} \| = (\sum_{m=1}^{3}(N_{i,j,k}[m])^2)^{\frac{1}{2}}$. The square brackets denote the components of the vector. Then we compute the intrinsic gradient, i.e., we project ∇u onto the plane normal to $\overrightarrow{N}$:

$(P_{\overrightarrow{N}}\overrightarrow{v})^n_{i,j,k} = \overrightarrow{v}^n_{i,j,k} - (\frac{\sum_{m=1}^{3} N_{i,j,k}[m]\cdot v_{i,j,k}[m]}{\| \overrightarrow{N}_{i,j,k}\|^2})\overrightarrow{N}_{i,j,k}$.

Finally, we use a backward-differences implementation of the divergence:

$\nabla_- \overrightarrow{w}_{i,j,k} = w_{i,j,k}[1] - w_{i-1,j,k}[1] + w_{i,j,k}[2] - w_{i,j-1,k}[2] + w_{i,j,k}[3] - w_{i,j,k-1}[3]$.

For the implementation of some PDE's like the flow visualization eq. (4.15), it might be convenient to alternate backward and forward differences for the computation of the 3D gradient and the

divergence. This way we avoid numerical artifacts. Since we have 3 dimensions, there are $2^3 = 8$ ways to compute the gradient or divergence, depending on which sort of difference scheme (backward B or forward F) we use for each dimension; the best way is to use *complementary* schemes in gradient and divergence (e.g., (B, F, B) for the gradient and (F, B, F) for the divergence), and change to other of the 8 possibilities in each iteration of the algorithm.

With forward time differences, the numerical implementation of the heat flow on implicit surfaces (eq. (4.7)) is then:

$u_{i,j,k}^{n+1} = u_{i,j,k}^{n} + \Delta t(\frac{1}{\| \overrightarrow{N}_{i,j,k} \|} \nabla_{-} \cdot ((P_{\overrightarrow{N}} \overrightarrow{v})_{i,j,k}^{n} \parallel \overrightarrow{N}_{i,j,k} \parallel))$.

If the embedding function is a signed distance function, obtaining eq. (4.8), this expression for the numerical implementation of the intrinsic heat flow is simplified even further, making it virtually trivial.

Bibliography

[1] F. Alouges, "An energy decreasing algorithm for harmonic maps," in J.M. Coron *et al.*, Editors, *Nematics*, Nato ASI Series, Kluwer Academic Publishers, Netherlands, pp. 1-13, 1991.

[2] L. Alvarez, P. L. Lions, and J. M. Morel, "Image selective smoothing and edge detection by nonlinear diffusion," *SIAM J. Numer. Anal.* **29**, pp. 845-866, 1992.

[3] L. Alvarez, F. Guichard, P. L. Lions, and J. M. Morel, "Axioms and fundamental equations of image processing," *Arch. Rational Mechanics* **123**, pp. 199-257, 1993.

[4] L. Ambrosio and M. Soner, "Level set approach to mean curvature flow in arbitrary codimension," *Journal of Differential Geometry* **43**, pp. 693-737, 1996.

[5] C. Ballester, M. Bertalmío, V. Caselles, G. Sapiro, and J. Verdera, "Filling-in by joint interpolation of vector fields and grey levels," *University of Minnesota IMA TR*, April 2000.

[6] M. Bertalmío, *Morphing Active Contours*, M.Sc. Thesis, I.I.E., Universidad de la Republica, Uruguay, June 1998.

[7] M. Bertalmío, G. Sapiro, and G. Randall, "Morphing active contours," *Proc. IEEE-ICIP*, Chicago, October 1998.

[8] M. Bertalmío, G. Sapiro, and G. Randall, "Region tracking on level-Sets methods," *IEEE Trans. on Medical Imaging* **18:5**, pp. 448-451, May 1999.

[9] M. Bertalmío, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," *Computer Graphics (SIGGRAPH)*, pp. 417-424, New Orleans, July 2000.

[10] A. Bertozzi *The mathematics of moving contact lines in thin liquid films*. Notices Amer. Math. Soc., Volume 45, Number 6, pp. 689-697, June/July 1998.

[11] M. Black, G. Sapiro, D. Marimont, and D. Heeger, "Robust anisotropic diffusion," *IEEE Trans. Image Processing* **7:3**, pp. 421-432, 1998.

[12] A. Blake and M. Isard, *Active Contours*, Springer-Verlag, New York, 1998.

[13] J. Bloomenthal, *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[14] C. Braverman. *Photoshop retouching handbook*. IDG Books Worldwide, 1998.

[15] H. Brezis, J. M. Coron, and E. H. Lieb, "Harmonic maps with defects," *Communications in Mathematical Physics* **107**, pp. 649-705, 1986.

[16] B. Cabral and C. Leedom. "Imaging vector fields using line integral convolution," *ACM Computer Graphics (SIGGRAPH '93)* **27:4**, pp. 263-272, 1993.

[17] V. Caselles, J. M. Morel, G. Sapiro, and A. Tannenbaum, Editors, "Special Issue on Partial Differential Equations and Geometry-Driven Diffusion in Image Processing and Analysis," *IEEE Trans. Image Processing* **7**, March 1998.

[18] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International Journal of Computer Vision* **22:1**, pp. 61-79, 1997.

[19] T. Chan and J. Shen, "Mathematical models for local deterministic inpaintings," *UCLA CAM TR* **00-11**, March 2000.

[20] Y. G. Chen, Y. Giga, and S. Goto, "Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations," *J. Diff. Geom.* **33**, 1991.

[21] S. Chen, B. Merriman, S. Osher, and P. Smereka, "A simple level set method for solving Stefan problems," *Journal of Computational Physics* **135**, pp. 8, 1995.

[22] L. T. Cheng, *The Level Set Method Applied to Geometrically Based Motion, Material Science, and Image Processing, PhD Thesis Dissertation, UCLA*, June 2000.

[23] R. Cohen, R. M. Hardt, D. Kinderlehrer, S. Y. Lin, and M. Luskin, "Minimum energy configurations for liquid crystals: Computational results," in J. L. Ericksen and D. Kinderlehrer, Editors, *Theory and Applications of Liquid Crystals*, pp. 99-121, IMA Volumes in Mathematics and its Applications, Springer-Verlag, New York, 1987.

[24] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators in nD," *Multi-res modeling group TR*, Caltech, September 2000 (obtained from www.multires.caltech.edu).

[25] U. Diewald, T. Preufer, and M. Rumpf, "Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces," *IEEE Trans. Visualization and Computer Graphics* **6**, pp. 139-149, 2000.

[26] J. Dorsey and P. Hanrahan, "Digital materials and virtual weathering," *Scientific American* **282:2**, pp. 46-53, 2000.

[27] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. "Multiresolution analysis of arbitrary meshes," *Computer Graphics (SIGGRAPH '95 Proceedings)*, pp. 173-182, 1995.

[28] M. Eck and H. Hoppe, "Automatic reconstruction of B-spline surfaces of arbitrary topological type," *Computer Graphics*, 1996.

[29] J. Eells and L. Lemarie, "A report on harmonic maps," *Bull. London Math. Soc.* **10:1**, pp. 1-68, 1978.

[30] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," *Proc. IEEE International Conference Computer Vision*, pp. 1033-1038, Corfu, Greece, September 1999.

[31] G. Emile-Male. *The Restorer's Handbook of Easel Painting*. Van Nostrand Reinhold, New York, 1976.

[32] L. C. Evans and J. Spruck, "Motion of level sets by mean curvature, I," *J. Diff. Geom.* **33**, 1991.

[33] O. Faugeras, F. Clément, R. Deriche, R. Keriven, T. Papadopoulo, J. Gomes, G. Hermosillo, P. Kornprobst, D. Lingrad, J. Roberts, T. Viéville, F. Devernay, "The inverse EEG and MEG problems: The adjoint state approach I: The continuous case," *INRIA Research Report* **3673**, June 1999.

[34] S. F. Frisken, R. N. Perry, A. Rockwood, and T. Jones, "Adaptively sampled fields: A general representation of shape for computer graphics," *Computer Graphics (SIGGRAPH)*, New Orleans, July 2000. '

[35] D. Gabor, "Information theory in electron microscopy," *Laboratory Investigation* **14**, pp. 801-807, 1965.

[36] F. Guichard and J. M. Morel, *Introduction to Partial Differential Equations in Image Processing*, Tutorial Notes, *IEEE Int. Conf. Image Proc.*, Washington, DC, October 1995.

[37] M. Grayson, "The heat equation shrinks embedded plane curves to round points," *J. Diff. Geom.* **26**, 1987.

[38] D. Heeger and J. Bergen. *Pyramid based texture analysis/synthesis*. Computer Graphics, pp. 229-238, SIGGRAPH 95, 1995.

[39] A. Hirani and T. Totsuka. *Combining Frequency and spatial domain information for fast interactive image noise removal*. Computer Graphics, pp. 269-276, SIGGRAPH 96, 1996.

[40] G. Huiskamp, "Difference formulas for the surface Laplacian on a triangulated surface," *Journal of Computational Physics* **95**, pp. 477-496, 1991.

[41] R. A. Hummel, "Representations based on zero-crossings in scale-space" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* **??**, pp. 204-209, IEEE New York, 1986.

[42] A. K. Jain, "Partial differential equations and finite-difference methods in image processing, part 1: Image representation," *J. of Optimization Theory and Applications* **23**, pp. 65-91, 1977.

[43] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision* **1**, pp. 321-331, 1988.

[44] C. Kenney and J. Langan. *A new image processing primitive: reconstructing images from modified flow fields*. University of California Santa Barbara Preprint, 1999.

[45] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Conformal curvature flows: from phase transitions to active vision," *Archive for Rational Mechanics and Analysis* **134**, pp. 275-301, 1996.

[46] B. B. Kimia, A. Tannenbaum, and S. W. Zucker, "Toward a computational theory of shape: An overview," *Lecture Notes in Computer Science* **427**, pp. 402-407, Springer-Verlag, New York, 1990.

[47] R. Kimmel, "Numerical geometry of images: Theory, algorithms, and applications," *Technion CIS Report* **9910**, October 1999.

[48] R. Kimmel, "Intrinsic scale space for images on surfaces: The geodesic curvature flow," *Graphical Models and Image Processing* **59**, pp. 365-372, 1997.

[49] D. King. *The Commissar Vanishes*. Henry Holt and Company, 1997.

[50] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," *Computer Graphics*, pp. 313-324, 1996.

[51] J. J. Koenderink, "The structure of images," *Biological Cybernetics* **50**, pp. 363-370, 1984.

[52] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner. *Detection of missing data in image sequences*. IEEE Transactions on Image Processing 11(4), 1496-1508, 1995.

[53] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner. *Interpolation of missing data in image sequences*. IEEE Transactions on Image Processing 11(4), 1509-1519, 1995.

[54] Landau, L. D., Lifshitz, E. M., 1959, *Fluid Mechanics*. Pergamon Press, Oxford, 1959.

[55] T. Lindeberg, *Scale-Space Theory in Computer Vision*, Kluwer, 1994.

[56] L. M. Lorigo, O. Faugeras, W. E. L. Grimson, R. Keriven, and R. Kikinis, "Segmentation of bone in clinical knee MRI using texture-based geodesic active contours," *Proceedings Medical Image Computing and Computer-Assisted Intervention, MICCAI '98*, pp. 1195-1204, Cambridge, MA, Springer, 1998.

[57] R. Malladi, J. A. Sethian and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE-PAMI* **17**, pp. 158-175, 1995.

[58] A. Marquina and S. Osher. *Explicit algorithms for a new time dependent model based on level set motion for nonlinear debluring and noise removal*. UCLA CAM Report 99-5, January 1999.

[59] S. Masnou and J.M. Morel. *Level-lines based disocclusion*. 5th IEEE Int'l Conf. on Image Processing, Chicago, IL. Oct 4-7, 1998.

[60] S. Mauch, "Closest point transform," www.ama.caltech.edu/∼seanm/software/cpt/cpt.html.

[61] B. Merriman, J. Bence, and S. Osher, "Motion of multiple junctions: A level-set approach,' *Journal of Computational Physics* **112**, pp. 334-363, 1994.

[62] B. Merriman, R. Caflisch, and S. Osher, "Level set methods, with an application to modeling the growth of thin films, *UCLA CAM Report* **98-10**, February 1998.

[63] O. Monga, S. Benayoun and O. Faugeras, "From partial derivatives of 3D density images to ridge lines," *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 354-359, 1992.

[64] H. P. Moreton and C. H. Séquin, "Functional minimization or fair surface design," *Computer Graphics (SIGGRAPH)*, pp. 167-176, 1992.

[65] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and variational problems," *Comm. Pure and App. Math.* **42**, 1989.

[66] W. J. Niessen, B. M. Romeny, and M. A. Viergever, "Geodesic deformable models for medical image analysis," *IEEE Trans. Medical Imaging* **17**, pp. 634-641, 1998.

[67] M. Nitzberg, D. Mumford, and T. Shiota, *Filtering, Segmentation, and Depth*, Springer-Verlag, Berlin, 1993.

[68] T. Ohta, D. Jasnow, and K. Kawasaki, "Universal scaling in the motion of random interfaces," *Physical Review Letters* **47**, pp. 1223-1226, 1982.

[69] S. Osher and L. I. Rudin, "Feature-oriented image enhancement using shock filters," *SIAM J. Numer. Anal.* **27**, pp. 919-940, 1990.

[70] S. J. Osher and J. A. Sethian, "Fronts propagation with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics* **79**, pp. 12-49, 1988.

[71] S. Osher, personal communication, October 1999.

[72] N. Paragios and R. Deriche, "A PDE-based level-set approach for detection and tracking of moving objects," *Proc. Int. Conf. Comp. Vision '98*, Bombay, India, January 1998.

[73] N. Paragios and R. Deriche, "Geodesic active regions for motion estimation and tracking," *Proc. Int. Conf. Comp. Vision*, Greece, September 1999.

[74] N. Paragios and R. Deriche, "Unifying boundary and region-based information for geodesic active tracking," *Proc. IEEE Conference on Computer Vision and Pettern Recognition*, Fort Collins, USA, June 1999.

[75] D. Peng, B. Merriman, S. Osher, H. Zhao, M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics* **155**, pp. 410-438, 1999.

[76] K. Perlin, "An image synthesizer," *Computer Graphics* **19**, pp. 287-296, 1985.

[77] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE-PAMI* **12**, pp. 629-639, 1990.

[78] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," *ACM Computer Graphics (SIGGRAPH)*, New Orleans, July 2000.

[79] C. B. Price, P. Wambacq, and A. Oosterlink, "Image enhancement and analysis with reaction-diffusion paradigm," *IEE Proc.* **137**, pp. 136-145, 1990.

[80] M. Proesmans, E. Pauwels, and L. van Gool, "Coupled geometry-driven diffusion equations for low-level vision," in [81].

[81] B. Romeny, Editor, *Geometry Driven Diffusion in Computer Vision*, Kluwer, The Netherlands, 1994.

[82] L. Rudin, S. Osher and E. Fatemi. *Nonlinear total variation based noise removal algorithms*. Physica D, 60, pp. 259-268, 1992.

[83] J. Serra, *Image Analysis and Mathematical Morphology,* Academic, New York, 1982.

[84] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis,* Cambridge University Press, Cambridge-UK, 2001.

[85] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences,* Cambridge University Press, Cambridge-UK, 1996.

[86] L. Simon, *Lectures on Geometric Measure Theory,* Australian National University, Australia, 1984.

[87] E. Simoncelli and J. Portilla. *Texture characterization via joint statistics of wavelet coefficient magnitudes*. 5th IEEE Int'l Conf. on Image Processing, Chicago, IL. Oct 4-7, 1998.

[88] M. Struwe, "On the evolution of harmonic mappings of Riemannian surfaces," *Comment. Math. Helvetici* **60**, pp. 558-581, 1985.

[89] B. Tang, G. Sapiro, and V. Caselles, "Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case," *Int. Journal Computer Vision* **36:2**, pp. 149-161, February 2000.

[90] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation," *IEEE Trans. PAMI* **13:11**, pp. 1115-1138, 1991.

[91] G. Taubin, "Estimating the tensor of curvature of a surface from a polyhedral approximation," *IEEE International Conference Computer Vision*, pp. 902-907, Boston, MA, 1995.

[92] P. Teo, G. Sapiro, and B. Wandell, "Creating connected representations of cortical gray matter for functional MRI visualization," *IEEE Trans. Medical Imaging* **16:06**, pp. 852–863, 1997.

[93] A. W. Toga, *Brain Warping*, Academic Press, New York, 1998.

[94] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control* **40** pp. 1528-1538, 1995.

[95] J. Tumblin and G. Turk, "LCIS: A boundary hierarchy for detail-preserving contrast reduction," Computer Graphics, pp. 83-90, SIGGRAPH 99, 1999.

[96] A. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society B* **237**, pp. 37-72, 1952.

[97] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion," *Computer Graphics* **25:4**, pp. 289-298, , July 1991.

[98] L. Vazquez, G. Sapiro, and G. Randall, "Segmenting neurons in electronic microscopy via geometric tracing," *Proc. IEEE ICIP*, Chicago, October 1998.

[99] S. Walden. *The Ravished Image*. St. Martin's Press, New York, 1985.

[100] J. Weickert, *Anisotropic Diffusion in Image Processing*, ECMI Series, Teubner-Verlag, Stuttgart, Germany, 1998.

[101] G. Winkenbach and D. H. Salesin, "Rendering parametric surfaces in pen and ink," *Computer Graphics (SIGGRAPH 96)*, pp. 469-476, 1996.

[102] A. P. Witkin, "Scale-space filtering," *Int. Joint. Conf. Artificial Intelligence* **2**, pp. 1019-1021, 1983.

[103] A. Witkin and P. Heckbert, "Using particles to sample and control implicit surfaces," *Computer Graphics (SIGGRAPH)*, pp. 269-278, 1994.

[104] A. Witkin and M. Kass, "Reaction-diffusion textures," *Computer Graphics (SIGGRAPH)* **25:4**, pp. 299-308, July 1991.

[105] A. Yezzi, S. Kichenassamy, P. Olver, and A. Tannenbaum, "Geometric active contours for segmentation of medical imagery," *IEEE Trans. Medical Imaging* **16**, pp. 199-210, 1997.

[106] G. Yngve and G. Turk, "Creating smooth implicit surfaces from polygonal meshes," *Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center. Georgia Institute of Technology*, 1999 (obtained from www.cc.gatech.edu/gvu/geometry/publications.html).

[107] X. Zeng, L. H. Staib, R. T. Schultz, and J. S. Duncan, "Segmentation and measurement of the cortex from 3D MR Images," *Proceedings Medical Image Computing and Computer-Assisted Intervention, MICCAI '98*, pp. 519-530, Cambridge, MA, Springer, 1998.

[108] D. Zhang and M. Hebert, "Harmonic maps and their applications in surface matching," *Proc. CVPR '99*, Colorado, June 1999.

[109] H. K. Zhao, T. Chan, B. Merriman, and S. Osher, "A variational level set approach to multiphase motion," *J. of Computational Physics* **127**, pp. 179-195, 1996.

[110] H. Zhao, S. Osher, B. Merriman, and M. Kang, "Implicit, non-parametric shape reconstruction from unorganized points using a variational level set method," *Comp. Vision and Image Understanding* **80**, pp. 295-314, 2000.