Active Contours
in Three Dimensions

Examensarbete utfört i Bildbehandling
vid Linköpings Tekniska Högskola

av

Jörgen Ahlberg

Reg nr: LiTH-ISY-EX-1708

Active Contours
in Three Dimensions

Thesis project done at Computer Vision Laboratory,
Linköping University

by

Jörgen Ahlberg

Reg nr: LiTH-ISY-EX-1708

Examiner: Klas Nordberg

Advisor: Johan Wiklund

                Computer Vision Laboratory
                Linköping University
                S - 581 83 Linköping
                Sweden

Linköping, September 1996

"Ready comprehension is often a knee-jerk response and the most dangerous form of understanding. *It blinks an opaque screen over your ability to learn. Be warned. Understand nothing. All comprehension is temporary."* [8]

ABSTRACT

To find a shape in an image, a technique called *snakes* or *active contours* can be used. An active contour is a curve that moves towards the sought-for shape in a way controlled by internal forces - such as rigidity and elasticity - and an image force. The image force should attract the contour to certain features, such as edges, in the image. This is done by creating an attractor image, which defines how strongly each point in the image should attract the contour.

In this thesis the extension to contours (surfaces) in three dimensional images is studied. Methods of representation of the contour and computation of the internal forces are treated.

Also, a new way of creating the attractor image, using the orientation tensor to detect planar structure in 3D images, is studied. The new method is not generally superior to those already existing, but still has its uses in specific applications.

During the project, it turned out that the main problem of active contours in 3D images was instability due to strong internal forces overriding the influence of the attractor image. The problem was solved satisfactory by projecting the elasticity force on the contour's tangent plane, which was approximated efficiently using sphere-fitting.

CONTENTS

1

INTRODUCTION

The project is here defined by giving scope and purpose. The outlines of the project and the thesis are also given.

1.1 BACKGROUND

In the late eighties it was suggested that it should be possible to follow edges in images by suggesting a curve (e.g., the circumference of an object) in an image, and then letting the curve itself move to a suitable shape and position. This curve should have physical properties like elasticity and rigidity, and also be attracted by edges in the image.

Such curves are called *active contours*, *deformable models* or *snakes* and have become popular especially in medical image analysis.

For the contour to be attracted to edges in the image an *energy image* or *attractor image* is created, which has high values where the original image has edges and low values otherwise. The attractor image gives the contour a potential energy by summing the energy in the points the contour passes.

The contour itself has an internal energy level determined by its shape (elasticity and rigidity) and by minimizing the total energy one aims at a smooth contour that follows the original image's edges well.

A natural extension is to segment three-dimensional images using active contours, and in several applications sequences of two-dimensional active contours are used (see e.g., [2] or [7]). In some cases one has also allowed the contour sequence to have elasticity and rigidity in the third dimension.

1.2 PURPOSE

The purpose of this work is to extend the theory and methods mentioned above to fit contours to shapes in 3D images. This includes finding good ways of representing the active contour as well as how to iterate and control the contour.

The second purpose is to examine how the attractor image can be created using the tensor representation of local orientation developed at Computer Vision Laboratory at Linköping University.

The active contour model should be implemented in the AVS[1] environment for easy use.

1.3 LIMITATIONS

- The debate on active contours mostly considers different numerical solutions (e.g., using finite element methods in the iteration). That is not at all treated in this thesis, since that alone would be enough work for a thesis in numerical analysis.

- It is not necessary to give the contour the properties elasticity and rigidity. Other possibilities have not been examined in this project.

1.4 ORDER OF WORK

The work has followed the following steps:

1. Literature studies.

   Searching for and reading what is written about active contours. Approximately twenty articles were found, of which maybe half a dozen ware directly used.

2. Implementation in 2D.

   To increase the understanding of and give an intuitive feeling of active contours and their behaviour in reality, implementing 2D active contours in MATLAB was the next step. This

---

1. *Application Visualization System* from *Advanced Visual Systems, Inc.*

1

also made it possible to experiment with different numerical solutions, attractor images, parameter functions, etc.

3. Representation.

   Developing ways of representing the contour in 3D and implementing the representations as classes in C++.

4. Attractor image.

   Finding out how a good attractor image can be created using the above mentioned tensor-representation and experimenting with different filters in the AVS environment.

5. Sensitive parameters.

   It was discovered that the contours are often unstable and very sensitive to change of parameters. This made it hard to control the contour, which made further studies necessary.

6. Implementation.

   Implementing the algorithms in C++, working on the previously created representation, and incorporating the code in AVS for visualization.

7. Experiments and evaluation.

   Performing experiments with different image volumes, attractor images and representations and evaluating the results.

1.5 OUTLINE

Chapter 2, "Active Contour Basics" describes the existing theory of active contours in two dimensions with some examples from the experiments performed early in the project.

Chapter 3, "The Attractor Image" describes how the attractor image can be created using estimates of local orientation, thus attracting the contour to planar structure in 3D images.

Chapter 4, "Representation in Three Dimensions" discusses the basic extension of the active contour model to three dimensions and describes suggestions of representation of the contour and computation of the internal forces.

Chapter 5, "Robust Control" treats the problem of instable contours, and gives suggestions of solutions.

Chapter 6, "Experiments" describes the experiments that were conducted and their results.

Chapter 7, "Conclusions & Future Work" contains, obviously, conclusions and some suggestions for future work and extensions of the active contour model.

1.6 ACKNOWLEDGEMENTS

2

ACTIVE CONTOUR BASICS

In this chapter some of the existing theory on active contours in two dimensions is described. A three-step process of fitting a contour to an image is described and examples are given.

2.1 WHAT IS AN ACTIVE CONTOUR?

The concept of *snakes* was first introduced in 1988 [9] and has later been developed by different researchers. The snake is the simplest form of active contours and the basic concept of this thesis.

A snake is a contour that can be described as a function $v : [0,1] \to \Re^2$ with some boundary conditions if required by the situation. The contour is placed on an image $f : \Re^2 \to \Re$, and it moves towards an "optimal" position and shape by minimizing its own energy.

Fitting active contours to shapes in images is an interactive process. The operator must suggest an initial contour, which is quite close to the intended shape. The contour will then be attracted to features in the image extracted by creating an attractor image.

Open and closed contours

The contour can be either a closed or an open curve. If the contour is open, one should take care to modify the contour's definition of its energy so that the end-points will not move in the same way as the other points (avoiding the contour dragging itself into itself and vanishing).

Balloons

An extension of the snake-concept called *balloons* is described in [1]. The idea is to add an inflation force to a closed contour to make the contour bypass irrelevant elements in the image. Using this inflation force, the contour initially suggested no longer needs to be in the neighbourhood of the optimal solution. For example one could place a small closed contour somewhere in the middle of a shape to which a contour should be fitted, and then inflate the contour to approximately the desired size. A variant of this is described in Chapter 5, "Robust Control".

2.2 THE ENERGY OF THE CONTOUR

The energy depends on the shape of the contour (internal energy) and on its positioning on the image according to

$$E(v, f) = E_{image}(v, f) + E_{int}(v) \qquad (2.1)$$

These energies influence all points along the contour with internal forces and an image force. When all forces are balanced, the total energy is at a minimum.

Internal energy

The internal energy of the contour depends on the shape of the contour and the parameter functions $\alpha(s)$ and $\beta(s)$ and is defined as

$$E_{int} = \int (\alpha(s) \cdot |v'(s)|^2 + \beta(s) \cdot |v''(s)|^2) ds \qquad (2.2)$$

The first term, $|v'(s)|^2$, will have larger values if there is a large gap between successive points on the contour and minimizing it will minimize the total length of the contour[1]. The second term, $|v''(s)|^2$, will be larger where the contour is bending and requires the contour to be as smooth as possible. These terms are weighted by parameter functions, and so $\alpha(s)$ determines the *elasticity* of the contour, and $\beta(s)$ determines the *rigidity*. If $\alpha(s)$ equals zero at some points then discontinui-

---

1. The equation (2.2) is a *membrane equation* known from mechanics (see e.g., [11]) combined with a rigidity-term.

ties are allowed there, and where $\beta(s)$ equals zero, discontinuous curvature such as corners are allowed.

To simplify, the parameter-functions will henceforth be regarded as constants.

Image energy

The image energy depends on how the contour is positioned on an *attractor image p*, and it is defined as

$$E_{image} = -\int p(v(s), f)ds \qquad (2.3)$$

where [9] uses an attractor image given by

$$p(x, f) = |\nabla f(x)|^2 \qquad (2.4)$$

This attractor image makes the contour attract to edges in the original image.

Minimizing the energy

Minimizing the energy (2.1) is equivalent to solving the corresponding Euler-Lagrange-equation

$$\alpha \cdot v'' - \beta \cdot v^{(4)} = -\nabla p(v, f) \qquad (2.5)$$

which simply means that the internal forces $\alpha \cdot v'' - \beta \cdot v^{(4)}$ shall balance the image force $\nabla p(v, f)$.

In practice one does not study the contour at all points. Instead, the contour is represented by a vector $\bar{v}$ of control points v_j. The control points must not be separated by more than a few pixels to prevent the contour from bypassing attractive but small areas in the image. It is the purpose of the elasticity force to keep the control points equidistant.

In the control points v_j the derivatives in (2.5) can be approximated by finite differences [5], which gives

$$v_j'' \approx v_{j-1} - 2v_j + v_{j+1} \qquad (2.6)$$

$$v_j^{(4)} \approx v_{j-2} - 4v_{j-1} + 6v_j - 4v_{j+1} + v_{j+2} \qquad (2.7)$$

Using these expressions (2.5) can be written as

$$A\bar{v} + \nabla p(\bar{v}, f) = 0 \qquad (2.8)$$

where A is the matrix given by (2.6) and (2.7) and insertion of α and β. A contour can then be calculated with iterations according to Euler's method [5] as

$$(I + A)\bar{v}_{k+1} = \bar{v}_k + \nabla p(\bar{v}_k, f) \qquad (2.9)$$

where $\bar{v}_0$ is the set of originally suggested control points.

One iteration makes each control point move so far in the direction of the image force that it is balanced by the resulting internal forces.

Since $I + A$ is pentadiagonal it is possible to solve the system (2.9) in *O(n)*-time.

2.3   THE CONTOUR-FITTING PROCESS

To find a suitably shaped and positioned contour three main steps are executed. As an example, the picture in Figure 2.1 is used, and a detailed description of how the example is constructed can be found in Appendix A, "Implementation in 2D".
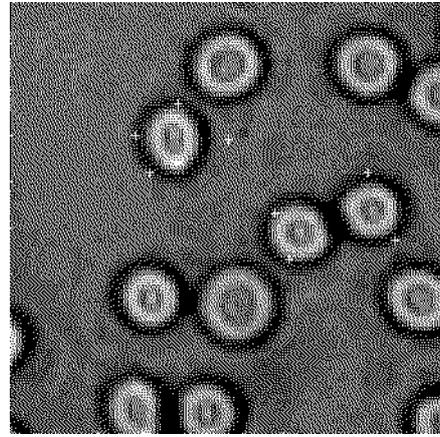
Figure 2.1: *The original image, $f(x)$, and the initial suggestions of control points (the white crosses).*

Step 1: Suggesting an initial contour

To make the contour attract to the shape in the image to which one wants to fit the contour, an initial suggestion should be given. As will be discussed in Section 4.3, "Representation", this can also include determining the representation of the contour. When segmenting a 2D image, the representation is straightforward; a vector of control points.

In the example in Figure 2.1, two closed contours are suggested, with four control points each. More control points are created automatically.

Step 2: Calculating the attractor image

The original image has to be filtered to create the attractor image according to (2.4). The attractor image is then preferably filtered with gradient filters to create the image force ∇p.

The image from Figure 2.1 is filtered with two 3×3 edge detecting filters (detecting edges in different directions) and in two scales. The

results are added, normalized, and squared, resulting in the attractor image in Figure 2.2.
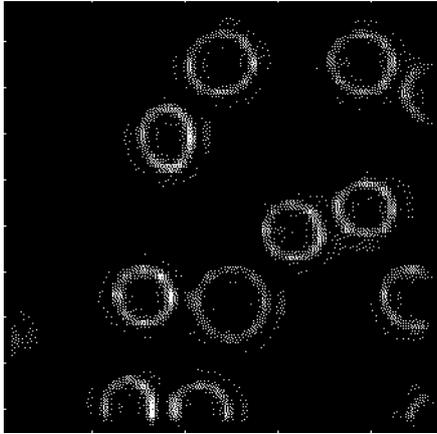
Figure 2.2: *The attractor image* $|\nabla f(x)|^2$ *.*

The process of creating the attractor image will be considered in Chapter 3, "The Attractor Image".

Step 3: Iterate

The suggested contour is iterated according to (2.9). The iteration will proceed until it eventually gives a stable contour or until the user interrupts the iteration.

In Figure 2.3, the two contours are shown after 50 iterations.
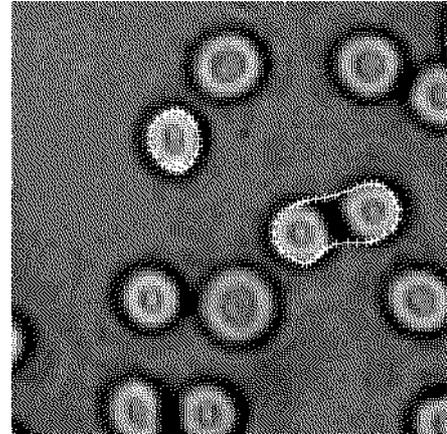
Figure 2.3: *The two contours after 50 iterations.*

Another example can be seen in Figure 2.4 where an open contour with fixed end-points is used to give a smooth segmentation of the image into two areas.
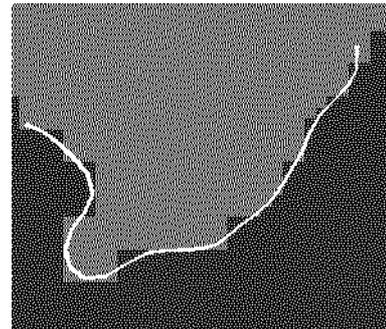
Figure 2.4: *Using an open contour to follow an edge.*

3

THE ATTRACTOR IMAGE

The attractor image has been calculated in approximately the same way in most applications. Since that model has some limitations, especially in 3D, some improvements based on orientation estimates are suggested here.

3.1 LIMITATIONS TO EARLIER MODELS

As mentioned in Chapter 2, the original attractor image in [9] was given by

$$p(x, f) = |\nabla f(x)|^2 \tag{3.1}$$

and the image force on the contour v in a point $s \in [0, 1]$ is given by $\nabla p(v(s), f)$. This attractor image has some limitations:

- It is only sensitive to edges, not to lines. That implies that the contour will be attracted to points on either side of a line. This is not always a problem, but in some applications it certainly is.

- In 3D, it is sensitive to lines as well as to planes. When trying to segment a volume in a 3D image, we are probably not interested in attracting the contour to linear structures in the image but want the volume to be bounded by planar structures.

The first problem is quite easily solved by taking the magnitude of an orientation estimate on the original image. This method is described in Section 3.3 below.

To solve the second problem it is suitable to create a local estimate of the probability of linear and planar structures, which is treated in Section 3.4.

3.2 THE ORIENTATION TENSOR

The theory of orientation tensors is described in detail in [6]. Only an informal explanation necessary for understanding the rest of the chapter will be given here.

A neighbourhood in an image (an n-dimensional signal) can be described with vectors pointing in the directions of signal variation. On an edge or a line that vector should be the edge/line normal. If the signal within the neighbourhood varies in more than one direction, e.g on a crossing of lines, several vectors might be needed.

However, a *simple* neighbourhood (varying in one direction only) can be expressed as[1]

$$f(x) = g(x^T \hat{s}) \tag{3.2}$$

where[2] $\hat{s}$ is a vector pointing in the direction of signal variation. Note that $f : \Re^n \to \Re$ but $g : \Re \to \Re$.

For simple signals we can express the local orientation with a tensor[3]

$$T = k \cdot \hat{s}\hat{s}^T \tag{3.3}$$

The tensor T will have the norm $\|T\| = k$, one non-zero eigenvalue k and the corresponding eigenvector $\hat{s}$. To this tensor we can add another tensor T_2, describing another simple signal with the orientation $\hat{r} \neq \hat{s}$. The resulting tensor will have two eigenvectors pointing in the two (orthogonal) directions in which the signal varies, and the corresponding eigenvalues tell the magnitude of the variation. The signal is still

---

1. This can also be written as $f(x) = g(x \cdot \hat{s})$ or $f(x) = g(cos(\angle(x, \hat{s})))$.
2. $\hat{s}$ denotes the normalized vector s, i.e., $\hat{s} = s / |s|$.
3. The tensor can be represented by a matrix everywhere in this thesis, and no special tensor theory is needed here. I will for the sake of consistency with other texts still call it a tensor.

constant in all directions orthogonal to the space spanned by the eigenvectors (or, equally, by $\hat{s}$ and $\hat{r}$).

Similarly, we can add any number of simple signals to the original one, building any signal, and the eigenvector corresponding to the largest eigenvalue will always point in the direction of the dominant orientation of the signal. If only one eigenvector exists, the signal varies in one direction and is a simple signal.

3.3 ATTRACTING TO LOCAL ORIENTATION IN 2D IMAGES

In two dimensions the local orientation can also be described as an orientation vector [6]

$$z = k \cdot \begin{bmatrix} s_1^2 - s_2^2 \\ 2 s_1 s_2 \end{bmatrix}, \ \hat{s} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \quad (3.4)$$

This vector has the same norm as the orientation tensor T and is consequently a measure of the magnitude of the local orientation, or the certainty that the neighbourhood is dominated by oriented structure. Note that the vector z does not point in the same direction as the dominant orientation of the neighbourhood, but uses a double-angle representation. This makes $f_1(x) = g(x^T \hat{s})$ and $f_2(x) = g(-x^T \hat{s})$ be represented by the same orientation vector.

By filtering an image with edge and line detectors in four directions, it is possible to estimate the orientation vector as follows:

If the convolution kernel $q_{line}(x)$ is a horizontal line detecting filter kernel and $q_{edge}(x)$ is a horizontal edge detecting filter kernel, then[1]

$$q_1(x, f) = \left| (f * q_{line})(x) \right. \quad (3.5)$$
$$\left. + i \cdot (f * q_{edge})(x) \right|$$

is an estimate of the dominance of local horizontal orientation in the image f. Combining q_1 with a "vertical estimate" q_3, one can get a more secure estimate in $q_1 - q_3$, since vertical and horizontal orientation are regarded as opposites. This is an estimate of the local one-dimensionality of the neighbourhood, i.e., that the neighbourhood contains structure oriented horizontally only (a negative value means that the neighbour-

hood is dominated by vertical structure). Using q_1, q_3 and their rotations by $\pi/4$ q_2 and q_4 the orientation vector can be created as

$$z = \begin{bmatrix} q_1 - q_3 \\ q_2 - q_4 \end{bmatrix} \quad (3.6)$$

An estimate of the local magnitude of orientation can be calculated as the norm of z, which equals

$$|z|^2 = \sum_{k=1}^{4} q_k^2 - 2(q_1 q_3 + q_2 q_4) \quad (3.7)$$

Using this norm

$$p(x, f) = |z(x, f)|^2 \quad (3.8)$$

as the attractor image will cause the contour to be attracted to oriented structure invariantly to phase, i.e., to lines as well as to edges.

As showed in [6] it is not really necessary with four complex[2] filters; in two dimensions it is quite enough using three filters[3] detecting lines/
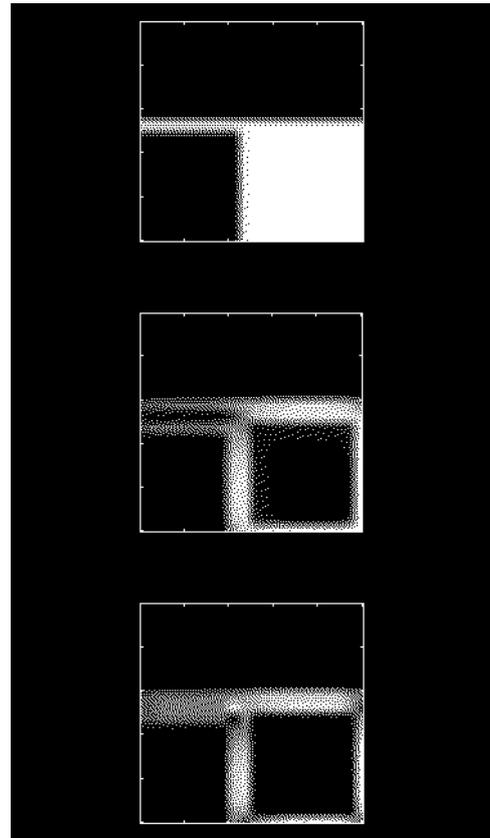
Figure 3.1: *The original image (top); the attractor image computed with the gradient method (middle); estimate of magnitude of local orientation (bottom).*

---

1. $(f * g)(x)$ denotes the function that is the result of convolving *f(x)* with *g(x)*.

edges in directions separated by $\pi/3$. The norm can then be calculated as

$$|z|^2 = \sum_{k=1}^{3} q_k^2 - q_1 q_2 + q_1 q_3 + q_2 q_3 \qquad (3.9)$$

What is important in order to get a good estimate, though, is that the filters together cover all directions.

In Figure 3.1 an image is filtered with the gradient method and the orientation method. Both methods uses four 3×3 filters (line detecting and line/edge detecting respectively) applied in two scales. The gradient method is the same as in the example in Section 2.3 and according to (3.1). The orientation method is used according to (3.7) and (3.8).

3.4 ATTRACTING TO PLANAR STRUCTURE IN 3D IMAGES

As described in [6] one can weight the line/edge filters mentioned above and estimate the orientation tensor T, whose eigensystem describes the local variation of the signal (or the local energy distribution in the Fourier domain) and therefore also the orientation.

Calling Ts eigenvalues λ_1, λ_2 and λ_3, where $\lambda_1 \geq \lambda_2 \geq \lambda_3$, and the corresponding eigenvectors e_1, e_2 and e_3, there are (in 3D) three characteristic cases:

- Plane case: $\lambda_1 \gg \lambda_2 \approx \lambda_3 \approx 0$

  The signal varies mainly in one direction, e_1, i.e the spatial neighbourhood describes a planar structure, with e_1 as normal vector (the plane is spanned by e_2 and e_3). The energy in the Fourier domain is mainly distributed along a line in the direction of e_1.

- Line case: $\lambda_1 \approx \lambda_2 \gg \lambda_3 \approx 0$

  The energy in the Fourier domain is distributed on a plane spanned by e_1 and e_2. The spatial neighbourhood is (nearly) constant in the same direction and describes a linear structure along e_3.

- Isotropic case: $\lambda_1 \approx \lambda_2 \approx \lambda_3$

  The neighbourhood has no orientation, i.e it varies equally in all directions.

Using a certainty estimate as attractor

In 3D, a signal containing planar structure is a simple signal. Thus, an estimate of planar structure is

$$\Delta(T) = \|T - T_1\|, \; T_1 = \lambda_1 \cdot e_1 e_1^T \qquad (3.10)$$

i.e how much the orientation tensor differs from the tensor corresponding to a simple signal in the direction of dominant orientation.

The spectrum theorem states that

$$T = \lambda_1 \cdot e_1 e_1^T + \lambda_2 \cdot e_2 e_2^T + \lambda_3 \cdot e_3 e_3^T \qquad (3.11)$$

and since $e_2 e_2^T \perp e_3 e_3^T$ the norm $\Delta(T)$ can be calculated as

$$\Delta(T) = \|T - T_1\| = \|\lambda_2 \cdot e_2 e_2^T + \lambda_3 \cdot e_3 e_3^T\| \qquad (3.12)$$

$$= \sqrt{\lambda_2^2 + \lambda_3^2}$$

$\Delta(T)$ will be close to zero in the presence of planar structure, and computation of (3.12) in each point in the image creates an attractor image that will attract the contour to planar structures in the image.

$\Delta(T)$ will also be close to zero when all eigenvalues are small. Consequently $\Delta(T)$ is very sensitive to noise on low-energy parts of the image, and therefore a division by λ_1 is suitable. The attractor image can then be expressed as

$$p_\Delta(x, f) = -\Delta_\lambda(eigenvalues(T(x, f))) \qquad (3.13)$$

where

$$\Delta_\lambda(\lambda) = \frac{\sqrt{\lambda_2^2 + \lambda_3^2}}{\lambda_1}, \; \Delta_\lambda : \Re^3 \to \Re \qquad (3.14)$$

---

2. Or eight real filters.
3. Actually, any number (equal or greater than three) of filters can be used. The estimation will be more secure the larger the number and the smaller the radial bandwidth of the filters. This decrease of bandwidth will however increase the spatial size of the filters, making the orientation estimate less local.
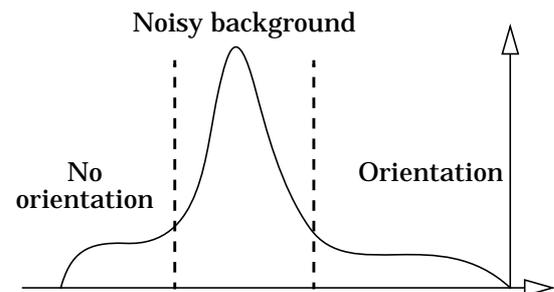
Figure 3.2: *Histogram of an attractor image.*

If the attractor image is calculated according to (3.13), the "typical" histogram look like Figure 3.2. Pixels within isotropic neighbourhoods will have lower values than the "oriented" pixels, but pixels within isotropic neighbourhoods dominated by noise (e.g., noise added to a black background) will have higher values (see Figure 3.2), i.e close to the oriented pixels.

Another certainty estimate

The interesting case of the cases mentioned above is the plane case, i.e., when $\lambda_1 >> \lambda_2 \approx \lambda_3$. Thus, the quota

$$\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \qquad (3.15)$$

is a certainty estimate of planar structure, which varies between one and one third. As with the first certainty estimate when all the eigenvalues are small, the quota (3.15) will be quite random. It is therefore suitable to multiply the quota with λ_1 to reduce the sensitivity to noise.
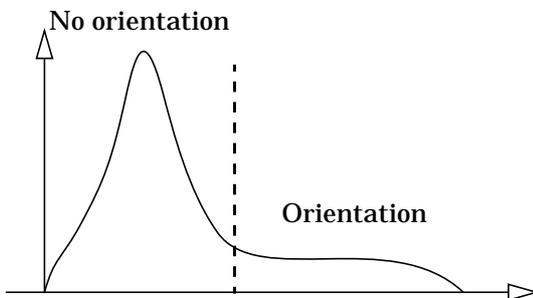
The attractor image used will consequently be expressed as

$$p_c(x, f) = c(eigenvalues(T(x, f))) \qquad (3.16)$$

where

$$c(\lambda) = \frac{\lambda_1^2}{\lambda_1 + \lambda_2 + \lambda_3}, \; c : \Re^3 \to \Re \qquad (3.17)$$

A typical histogram of the attractor image computed according to (3.16) will look like Figure 3.3. The oriented and isotropic areas seems not to be as well separated as with (3.13), but this method is not as sensitive to noise. The choice of method should therefore depend on how noisy the image is around the object one wishes to segment.

3.5 CREATING THE ATTRACTOR IMAGE

As indicated above, several steps are required to create the attractor image and the resulting image force. The sequence of these steps is illustrated in Figure 3.4 and described below (a 3D-image is assumed).

1. Tensor filtering

   Filter the image with at least six edge/line-detectors and create the orientation tensor according to Section 3.4.

2. Averaging

   The tensor field usually needs some averaging (low-pass-filtering) to reduce randomness of the orientation caused by noise. An ordinary $5 \times 5 \times 5$ or $7 \times 7 \times 7$ gaussian kernel is used.

3. Computing eigenvalues

   Compute the eigenvalues of the tensor field and sort them in decreasing order.

4. Detecting planar structure

   Compute the energy according to equation (3.16) or (3.13).

5. Thresholding

   The parts of the original image without dominant local orientation, e.g black background, will (especially in the presence of noise) contribute to the calculated attractor, however with quite low values. It is, though, preferable to set all these points to zero. This will straighten out the contour where crossing such regions, since the contour will there only be under the influence of the internal forces (elasticity and rigidity).

   The typical histogram (see Figure 3.3) is mainly divided into two ares; all uninteresting pixels ("isotropic case") have low values and the pixels with planar structure have higher values. Setting all pixels in the "isotropic area" to zero and subtracting this value from the other pixels results in a more reliable attractor image.

Figure 3.3: *Histogram of an attractor image.*

If the second certainty estimate is used, then a (variable) threshold of slightly more than $\lambda_1/3$ will set all "isotropic pixels" to zero.

Original image

↓

| Tensor filtering |

↓

| Averaging |

↓

| Computing eigenvalues |

↓

| Detecting planar structure |

↓

| Thresholding |

↓

| Gradient filtering |

↓
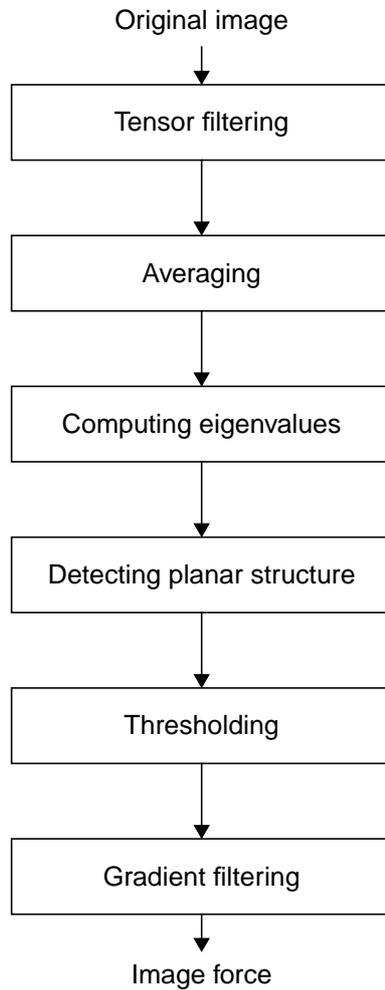
Image force

Figure 3.4: *The process of calculating the image force.*

6. Gradient filtering

Computing the 3D-gradient of the attractor image will result in the image force used when iterating the contour. The image force is a function $f : \Re^3 \to \Re^3$, i.e., every "pixel" in the 3D-image is a three-element vector.

4

---

*R*EPRESENTATION IN *T*HREE *D*IMENSIONS

This chapter treats the basic extension of the active contour model from two to three dimensions. Different representations of 3D contours are described, which also makes it necessary to revisit the methods of calculating the internal forces.

4.1 THE 3D CONTOUR

A 3D contour is typically described by a function $v : [0,1] \times [0,1] \to \Re^3$. The contour is placed on an image $f : \Re^3 \to \Re$ and is moved by iteration in the same way as in the 2D case.

The simplest way to create an active contour in 3D is by using repeated 2D contours. This method is often used since it is also quite intuitive in many applications with image sequences. The major drawback is that it is impossible to vary the distance between the control points in the third dimension, so that exactly one point per pixel (in the third dimension) is required.

In some applications (e.g., [2] and [7]) one does not even care about elasticity and rigidity in the third dimension and calculates a number of independent contours on separate images. This, of course, can result in very unpleasant contours, e.g., if a piece of a line through the image sequence is missing in one or some images.

In this thesis, "true" 3D contours are discussed, i.e., the control points can move in three directions and both the external and internal forces operate in 3D.

4.2 THE ENERGY OF THE CONTOUR

In 3D the image force is defined in the same way as in 2D, but the internal energy has to be calculated in a slightly different way, which also leads to a modification of the Euler-Lagrange-equation (2.5).

To calculate the internal energy in 3D, some additional parameter functions are required, and the energy is now defined as (with $v = v(s, r)$)

$$E_{int} = \int (\alpha_s |v_s'|^2 + \alpha_r |v_r'|^2 + \beta_s |v_{ss}''|^2 \qquad (4.1)$$
$$+ \beta_r |v_{rr}''|^2 + \beta_{sr} |v_{sr}''|^2) ds dr$$

where α_s and α_r determine the elasticity along the s- and r-axis respectively, β_s and β_r determine the corresponding rigidities and β_{sr} determines the resistance to twist.

The energy is minimized by finding the v that satisfies the Euler-Lagrange-equation

$$\nabla p(v, f) + \alpha_s v_{ss}'' + \alpha_r v_{rr}'' \qquad (4.2)$$
$$- \beta_s v_{ssss}^{(4)} - \beta_r v_{rrrr}^{(4)} - \beta_{sr} v_{ssrr}^{(4)} = 0$$

which, as in 2D, means that the internal forces shall balance the image force.

4.3 REPRESENTATION

A 2D contour is simply represented by a vector of control points where neighbouring control points are represented by neighbouring elements in the vector. (This is what causes the matrix **A** in (2.8) - (2.9) to be pentadiagonal.)

In 3D, the contour is most easily represented by a matrix of control points, a mesh. The mesh is not suitable in all cases; this will be discussed below.

Representing the contour with a mesh

The two different types of 2D contours (open and closed contours) can with the corresponding boundary conditions

$$v(s, 0) = v(s, 1) \forall s \tag{4.3}$$

$$v(0, r) = v(1, r) \forall r \tag{4.4}$$

be extended to a cylinder if one of (4.3) and (4.4) is met or a torus if both (4.3) and (4.4) are met. If none of the conditions are met, the contour is just an open surface.

     Assuming an $n \times m$-mesh, it is trivial to expand the mesh to an $n(n - a) \times m(m - b)$-mesh, where a and b equal one if equation (4.3) and (4.4) respectively are satisfied (and zero otherwise).

Representing the contour with a polyhedron

To create a sphere-like contour one can use the following set of boundary conditions:

$$v(s, 0) = v(s, 1) \forall s \tag{4.5}$$

$$v(0, r_1) = v(0, r_2) \forall r_1, r_2 \tag{4.6}$$

$$v(1, r_1) = v(1, r_2) \forall r_1, r_2 \tag{4.7}$$

     However the control points will then be placed on the sphere in a quite irregular manner (with regard to their mutual distances), and there is no obvious way to modify (4.1) to fit this topology. Consequently the mesh-representation is not suitable for spherical contours - instead it is preferable to place the control points regularly on a sphere, i.e., to let the control points be the vertices of a regular polyhedron (in case of eight control points, they should be placed like the corners of a cube). Unfortunately there is no regular
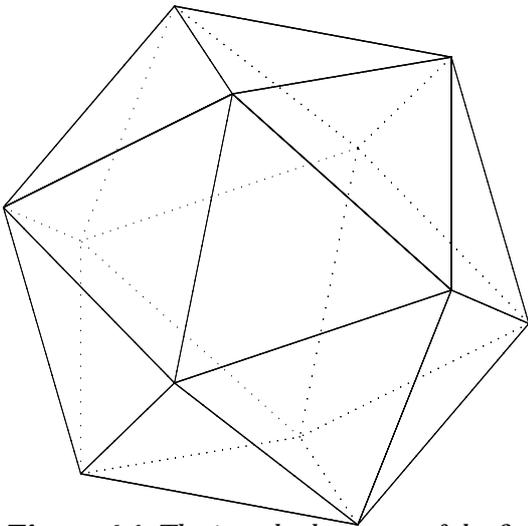
Figure 4.1: *The icosahedron, one of the five Platonic polyhedra.*

polyhedron with more than twenty vertices, as proved by Plato some time ago [4].

     A solution is to create a quasi-regular polyhedron with equal distances between each vertex and its neighbours. Such polyhedra can be created recursively from an icosahedron, see Figure 4.1, using the following algorithm:

1. Place a new vertex at the midpoint of each edge.

2. Remove the old edges and create new edges between all vertices with a mutual distance that equals half of the distance between the old vertices.

3. Project the new vertices on the circumsphere of the polyhedron.

The original twelve vertices have five neighbours while all new vertices will have six neighbours. The number of faces, edges and vertices (F_n, E_n and V_n where n is the number of recursions) grow quite quickly. The original polyhedron P_0 has got the characteristics $(F_0 \, E_0 \, V_0) = (20 \; 30 \; 12)$, and the characteristics of polyhedron P_n is calculated recursively (directly from the algorithm) as

$$\begin{cases} F_n = 4 \cdot F_{n-1} \\ E_n = 2E_{n-1} + 3F_{n-1} \\ V_n = V_{n-1} + E_{n-1} \end{cases} \tag{4.8}$$

Expanding E_n as

$$E_n = 2(2E_{n-2} + 3F_{n-2}) + 3F_{n-1} \tag{4.9}$$

$$= 2(2(2E_{n-3} + 3F_{n-3}) + 3F_{n-2}) + 3F_{n-1}$$

(and so on) makes it obvious that E_n can be written as a closed expression:

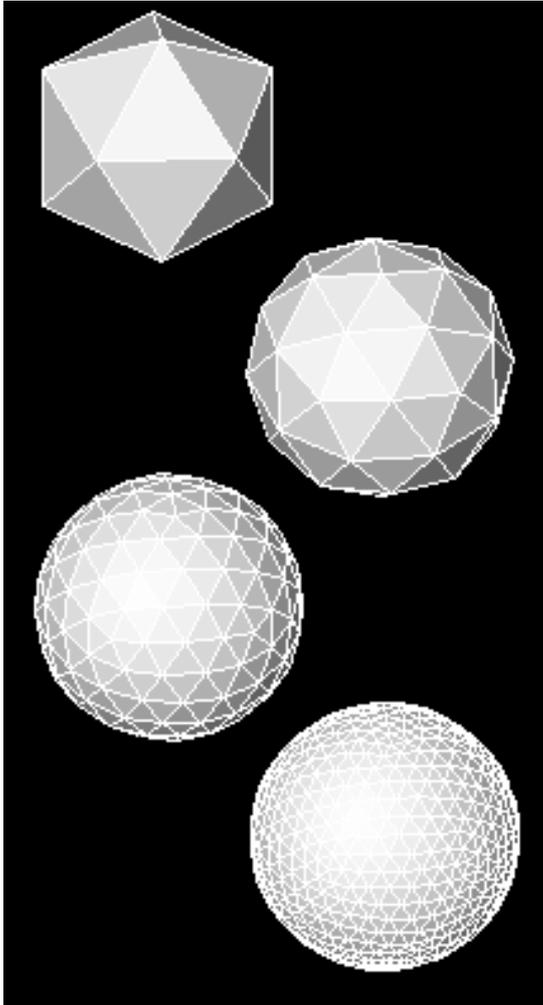$$E_n = 2^n E_0 + 3 \cdot \sum_{k=0}^{n-1} 2^{n-1-k} \cdot F_k \tag{4.10}$$

$$= 2^n E_0 + 3 \cdot 2^{n-1} \cdot \sum 2^{-k} \cdot F_0 \cdot 4^k$$

$$= 2^n E_0 + 3 \cdot 2^{n-1} \cdot F_0 \cdot \sum 2^k$$

$$= 30 \cdot 4^n$$

In a similar way V_n can be expanded and expressed as

$$V_n = V_0 + \sum_{k=0}^{n-1} E_k \qquad (4.11)$$

$$= 12 + 30 \cdot \sum 4^k$$

$$= 2 + 10 \cdot 4^n$$

F_n is trivially rewritten, and so (4.8) equals

$$\begin{cases} F_n = 20 \cdot 4^n \\ E_n = 30 \cdot 4^n \qquad (4.12) \\ V_n = 2 + 10 \cdot 4^n \end{cases}$$


Figure 4.2: *The icosahedron and its first three expansions.*

For example, P_3 and P_4 have *642* and *2562* vertices respectively, which could be suitable amounts in practical applications. The four first polyhedra (P_0 through P_3) are illustrated in Figure 4.2.

4.4 THE INTERNAL FORCES INDEPENDENT OF REPRESENTATION

Elasticity can be regarded as a dragging force from each of the neighbouring control points, i.e., the elasticity force in a control point is simply the vector from the control point to the average of the neighbouring control points (see Figure 4.3). In 2D, this is the same as equation (2.6).
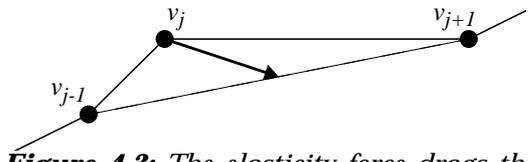
Figure 4.3: *The elasticity force drags the control point v_j towards the average of the control points v_{j-1} and v_{j+1}.*

Rigidity can be regarded as the force from a control point to a point linearly "predicted" by the two "earlier" points, as shown in Figure 4.4, i.e., the vector from v_{j-2} to v_{j-1} is added to v_{j-1}, and v_j is forced towards the resulting point. This means that the rigidity force in a point v_j is $2v_{j-1} - v_{j-2} - v_j$. The "prediction" can be made in all directions and be generalized to higher dimensionality.
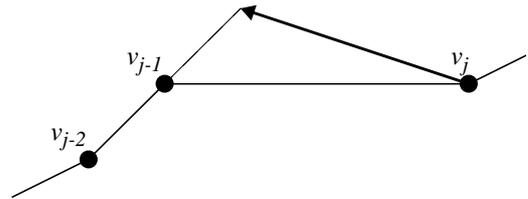
Figure 4.4: *The rigidity force drags the control point v_j towards the position predicted by the control points v_{j-2} and v_{j-1}. The control points v_{j+2} and v_{j+1} creates a similar force.*

This view of rigidity is not the same as the physical property; it will not straighten lines by suppressing corners but by raising neighbouring points to the same level as the corner. By adding twice the elasticity to the rigidity it becomes the same.

This can more formally be expressed in the following way:

Let $P_{j,d} = \{p_{j,d,m}\}$, $m = 1, ..., M$, be the set of M points on a distance d steps from a control point v_j. By setting $c_{j,d}$ to the average of all points on the distance d from the point v_j, i.e.,

$$c_{j,d} = \sum_m \frac{p_{j,d,m}}{|P_{j,d}|} \qquad (4.13)$$

the elasticity and rigidity forces in the point v_j can be expressed as

$$F_{elast}(v_j) = c_{j,1} - v_j \qquad (4.14)$$

$$F_{rigid}(v_j) = 4c_{j,1} - c_{j,2} - 3v_j \qquad (4.15)$$

These equations are applicable on a contour in 2D as well as on a mesh (describing a surface, torus or cylinder) or a polyhedron. Their best use is however on the polyhedron, where it is not obvious how to calculate the s- and r-derivatives. They also make it possible to choose whether to use *4*-connectivity or *8*-connectivity in the mesh.

The elasticity and rigidity parameters are then again only two - α (for elasticity in all directions) and β (for rigidity in all directions) - and the sought-for balance of the contour forces is now expressed as

$$\alpha \cdot F_{elast}(\bar{v}) + \beta \cdot F_{rigid}(\bar{v}) + \nabla p(\bar{v}, f) = 0 \qquad (4.16)$$

where $\bar{v}$ is the set of control points representing the contour v. If $\bar{v}$ is a vector (4.16) equals (2.5), and if $\bar{v}$ is a mesh with 4-connectivity (4.16) equals (4.2) with $\alpha_s = \alpha_r$, $\beta_s = \beta_r$ and $\beta_{sr} = 0$.

Robust Control

This chapters examines the problem of instability due to internal forces, which are too strong but which are still too weak to fulfil their purpose. The suggested solution limits the internal forces' influence on the shape of the contour, instead letting them influence the point distribution only. This also makes it possible to use the balloon-concept in a stable way.

5.1 The Problem of Sensitive Parameters

The purpose of the elasticity force is to keep the control points equidistant along the contour. However, if the distance between the control points is too large on an overly curved image contour, the elasticity force tends to drag the control points away from rather than along the image contour; see Figure 5.1.

This can be avoided by decreasing the values of the elasticity parameter function $\alpha(s)$, but a too weak elasticity force (compared to the image force) makes the points gather in strong areas on the attractor image in a very non-equidistant manner. To find an $\alpha(s)$ (and a $\beta(s)$) that solves the problem, it may be necessary to set its values individually in each point, but even then it is not certain that a solution exists.

5.2 Outline of Solution

Xu et al. suggest in [14] that the tangent of the contour should be approximated at each control point, and that a pressure force perpendicular to the tangent should be added. This force has in each control point the same magnitude as the part of the internal force that is perpendicular to the tangent, but the two forces shall point in opposite directions and consequently balance each other. As a result the internal forces may only move a control point along the direction of the tangent.

A simpler way to express the same thing is that the elasticity force should be projected on the tangent plane of the contour before influencing the contour.

The problem is then reduced to approximating the tangent, or, in 3D, the tangent plane, in each point. Two methods are described below.
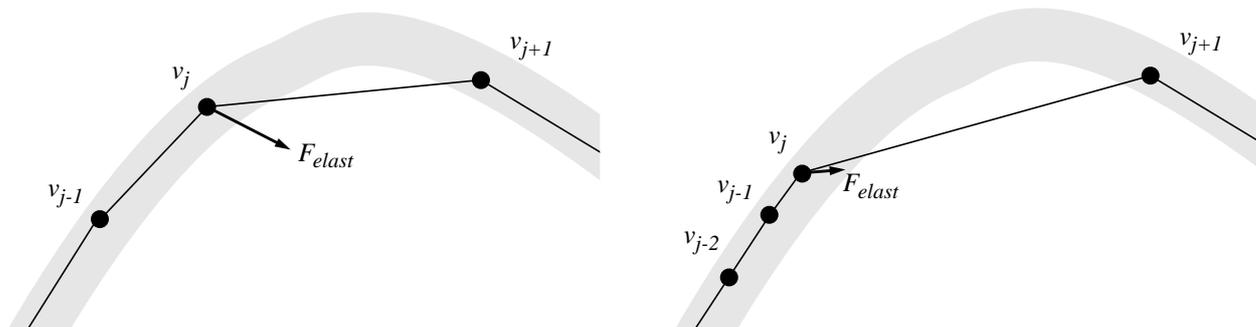
Figure 5.1: An overly strong elasticity force, F_{elast}, drags the control point v_j away from the image contour (left), while a weak elasticity force cannot keep the points equidistant (right).

A comment on complexity of algorithms

Only computational complexity (complexity in time) is considered here, since the complexity in space (i.e., memory) is in any case very small compared to the space needed for the attractor image.

The computational complexity is approximated by the number of floating point operations needed. Fixed point operations for program control, etc., are regarded as taking zero time.

5.3 RECONSTRUCTING A CONTOUR USING BASIS FUNCTIONS

If the contour is represented by a mesh, we can regard the control points as a regular sampling of a signal. We can then reconstruct the signal as a linear combination of a set of basis functions with the values in the sampling points as coefficients.

Assuming the signal is limited to a finite range/area/volume in the Fourier domain, a perfect reconstruction is possible. For a signal to be band-limited, it must be unlimited in space, and therefore only closed contours (periodic signals) can be band-limited.

Requirements on basis functions

For a set of functions to be used as basis functions the following requirements have to be met:

- The functions should equal zero in all sampling points but the one corresponding to the basis function.

- The sum of all basis functions should equal one in *all* points (not just in the sampling points).

- The functions should belong to a Sobolev-space of the fourth order, i.e., all derivatives of up to the fourth order should be squared integrable. If not, the rigidity force does not necessarily converge.

Choice of basis functions

If the contour is closed in a dimension (i.e., equation (4.3) or (4.4) in Section 4.3 is satisfied), we can use a band-limited periodic function as the basis in that dimension. Its period should equal the length of the contour, and a suitable basis function (if the number of samples is odd) is

$$\varphi_n(x) = \frac{1}{n}\left(1 + 2 \cdot \sum_{k=1}^{(n-1)/2} cos\left(\frac{2k\pi x}{n}\right)\right) \quad (5.1)$$

where n is the number of samples and the other basis functions in the set are the translations $\varphi_n(x-k)$. The derivative function of the m:th order is then given by

$$\varphi_n^{(m)}(x) = \frac{2}{n} \cdot \sum_{k=1}^{(n-1)/2} \left(\frac{2k\pi}{n}\right)^m cos\left(\frac{2k\pi x}{n} - \frac{m\pi}{2}\right) \quad (5.2)$$

In case of an open contour it is not possible to use band-limited functions. Instead the open contour can be regarded as a small part of a much larger (closed) contour, whose sampled values all equal zero outside the limited part.

For example, (5.1) can be used as a basis function, but with an n *at least* twice the number of samples (in order for two points to correlate less as the distance between them grows).

Reconstruction of the contour and its derivatives

The contour $v : \Re^2 \to \Re^3$ can be written as

$$v(s_1, s_2) = \left[x_1(s_1, s_2)\ x_2(s_1, s_2)\ x_3(s_1, s_2)\right]^T \quad (5.3)$$

with the derivatives

$$\frac{\partial^m v}{\partial s_k^m} = \left[\frac{\partial^m x_1}{\partial s_k^m}\ \frac{\partial^m x_2}{\partial s_k^m}\ \frac{\partial^m x_3}{\partial s_k^m}\right]^T \quad (5.4)$$

Assuming the contour to be represented by three $n_1 \times n_2$-meshes $\bar{x}_1$, $\bar{x}_2$ and $\bar{x}_3$ the derivatives in a control point (s_1, s_2), $s_j \in \{0, \dots, n_j - 1\}$, equal

$$\begin{cases} \dfrac{\partial^m x_j}{\partial s_1^m} = \displaystyle\sum_{k=0}^{n_1-1} \bar{x}_j(k, s_2) \cdot \varphi_{n_1}^{(m)}(s_1 - k) \\ \dfrac{\partial^m x_j}{\partial s_2^m} = \displaystyle\sum_{k=0}^{n_2-1} \bar{x}_j(s_1, k) \cdot \varphi_{n_2}^{(m)}(s_2 - k) \end{cases} \quad (5.5)$$

The values of the functions $\varphi_{n_j}^{(m)}(x-k)$ can be calculated in advance (i.e outside the iteration-loop) and the values stored in vectors

$$\bar{\varphi}_n^{(m)}(s) = \begin{bmatrix} \varphi_n^{(m)}(s) \\ | \\ \varphi_n^{(m)}(s-(n-1)) \end{bmatrix} \quad (5.6)$$

Using (5.6) and the matrix given by

$$\bar{x}_{s_1}(s_2) = \begin{bmatrix} \bar{x}_1(0, s_2) & \dots & \bar{x}_1(n_1 - 1, s_2) \\ \bar{x}_2(0, s_2) & \dots & \bar{x}_2(n_1 - 1, s_2) \\ \bar{x}_3(0, s_2) & \dots & \bar{x}_3(n_1 - 1, s_2) \end{bmatrix} \qquad (5.7)$$

(and the corresponding $\bar{x}_{s_2}(s_1)$) the derivatives can be expressed as

$$\begin{cases} \dfrac{\partial^m}{\partial s_1^m} v(s_1, s_2) = \bar{x}_{s_1}(s_2) \bar{\varphi}_{n_1}^{(m)}(s_1) \\[2mm] \dfrac{\partial^m}{\partial s_2^m} v(s_1, s_2) = \bar{x}_{s_2}(s_1) \bar{\varphi}_{n_2}^{(m)}(s_2) \end{cases} \qquad (5.8)$$

Since the values of the basis functions can be calculated in any point, it is possible to interpolate the contour and its derivatives in any point (s_1, s_2). If neither of s_1 or s_2 are integer values, the contour $v^{(m)}$ has first to be interpolated in all the points $(s_1, 0) \dots (s_1, n_2 - 1)$ (using the second equa-

tion in (5.8)). Then $v^{(m)}(s_1, s_2)$ can be reconstructed using the first equation in Figure 5.8.

The tangent plane

The tangent plane of the contour is given by its normal vector

$$\mu = \frac{\partial v}{\partial s_1} \times \frac{\partial v}{\partial s_2} \qquad (5.9)$$

which by insertion of (5.8) can be expressed as

$$\mu(s_1, s_2) = \bar{x}_{s_1}(s_2) \bar{\varphi}_{n_1}^{(m)}(s_1) \times \bar{x}_{s_2}(s_1) \bar{\varphi}_{n_2}^{(m)}(s_2) \qquad (5.10)$$

Computational complexity

Computing (5.8) requires $6(n_1 + n_2 - 1)$ floating point operations. Another 9 operations are needed to calculate the normal vector μ, so a total of $6(n_1 + n_2) + 3$ operations are needed to calculate μ in one point, i.e in $O(n_1 + n_2)$ time.
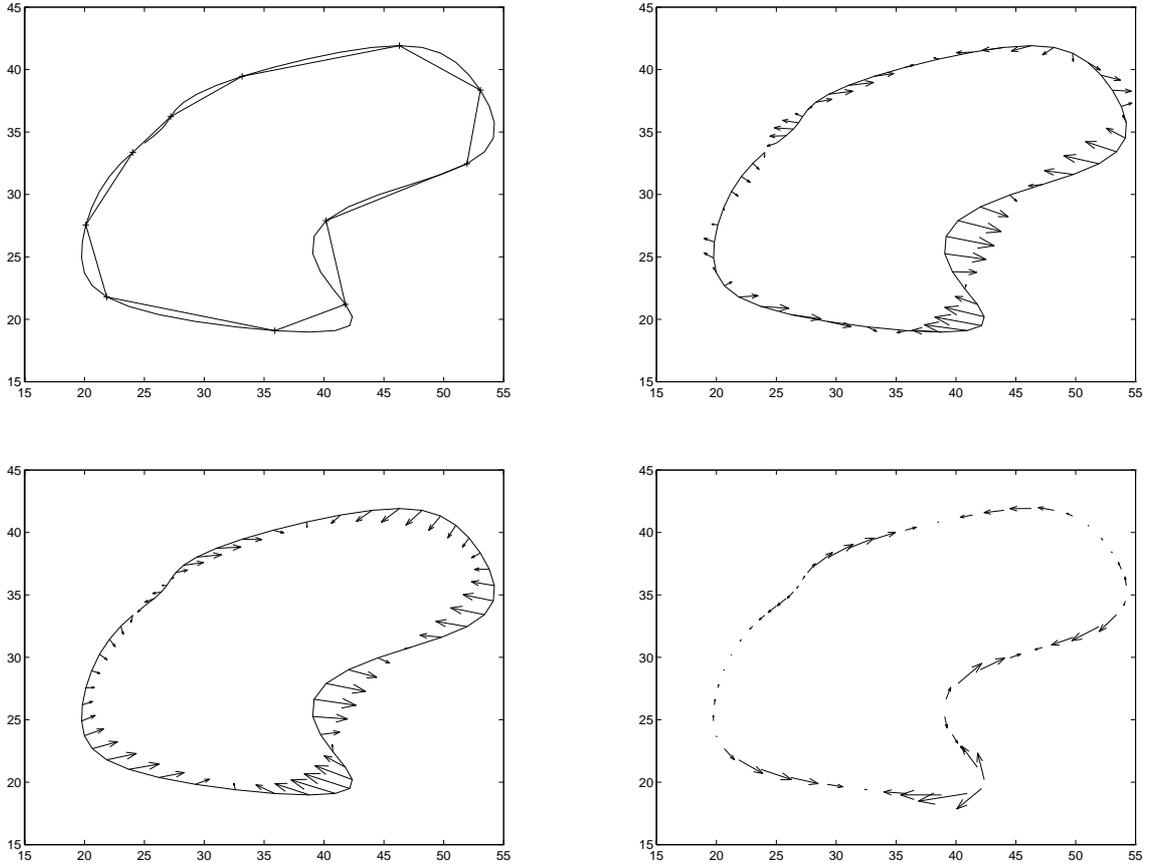
Figure 5.2: The contour represented by 11 control points and its reconstruction (top left); the rigidity force (top right); the elasticity force (bottom left); the elasticity force projected on the tangent in each point (bottom right).

Global internal forces

Equation (5.8) can be also used to calculate the elasticity and rigidity forces. Compared to using the earlier methods (e.g., equations (4.14) - (4.15) in Section 4.4) this will increase computation complexity by a factor proportional to $n_1 + n_2$. As a positive effect the internal forces will behave globally, i.e., if when a control point is moved, the force on all other points will be immediately effected, instead of several iterations later.

An example

In Figure 5.2, a 2D-contour with 11 control points is interpolated in 55 points using the basis function (5.1). Its second and fourth derivatives, i.e the elasticity and the rigidity force respectively, are interpolated as well.

5.4 APPROXIMATING THE TANGENT PLANE WITH SPHERES

On a generalized representation of the contour, e.g., the polyhedron-representation described in Chapter 4, the method described above cannot be used (unless a way of expressing basis functions for a general polyhedron can be found). Instead, an approximating method will be used.

In [14] it is suggested that a 2D contour can be approximated with tangent arcs, i.e., the tangent in a point v_j is approximated with the circle that passes through points v_{j-1}, v_j and v_{j+1}. The tangent is represented by its normal vector $\hat{\mu}_j$, pointing from v_j towards the centre of the circle m_j; see Figure 5.3.
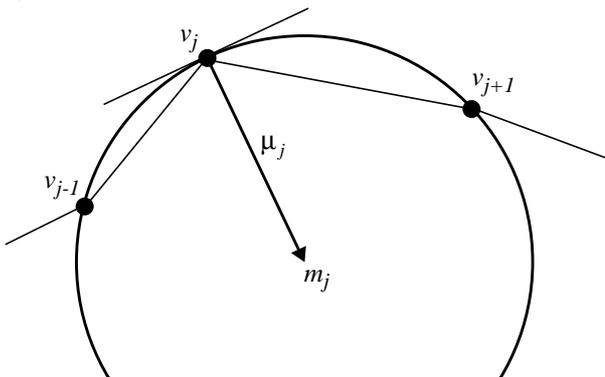

Figure 5.3: The tangent approximated with the circle through three control points.

By definition m_j must be situated at the same distance from v_{j-1}, v_j and v_{j+1}, which implies that μ_j projected on v_{j-1} - v_j and v_{j+1} - v_j (see Figure 5.3)

will reach half-way along these vectors respectively, i.e μ_j should satisfy

$$\begin{cases} \left(\dfrac{v_{j-1} - v_j}{\|v_{j-1} - v_j\|}\right) \cdot \mu_j = \dfrac{1}{2}\|v_{j-1} - v_j\| \\[3mm] \left(\dfrac{v_{j-1} + v_j}{\|v_{j-1} + v_j\|}\right) \cdot \mu_j = \dfrac{1}{2}\|v_{j-1} + v_j\| \end{cases} \quad (5.11)$$

Extending to three dimensions

In 3D, we use three neighbouring points, $v_{j,a}$, $v_{j,b}$ and $v_{j,c}$, and calculate the sphere which has all four points as edge points. In the same way as (5.11), the sought-for μ_j is found by solving the system

$$\begin{bmatrix} (v_{j,a} - v_j)^T \\ (v_{j,b} - v_j)^T \\ (v_{j,c} - v_j)^T \end{bmatrix} \mu_j = \frac{1}{2} \begin{bmatrix} \|v_{j,a} - v_j\|^2 \\ \|v_{j,b} - v_j\|^2 \\ \|v_{j,c} - v_j\|^2 \end{bmatrix} \quad (5.12)$$

Spheres approximating a polyhedron

Using the suggested polyhedron representation (Section 4.3), where most points have six neighbours, several different tangent planes can be calculated. Preferably three neighbouring points that are not neighbouring each other are used to calculate a normal $\mu_{j,1}$. The three neighbouring points not used can be used (see Figure 5.4) to calculate another normal $\mu_{j,2}$, and the average (after normalization) can be used.
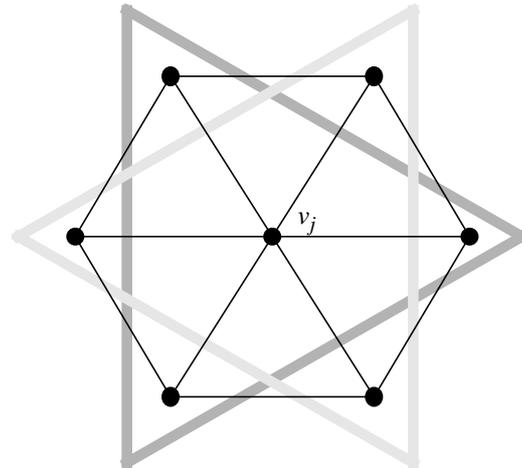

Figure 5.4: The control point v_j and its six neighbours. The two gray triangles surround the points that should be used for calculating $\mu_{j,1}$ and $\mu_{j,2}$ respectively.

Some care is needed doing the averaging. If the two normal vectors are related like

$\hat{\mu}_{j,1} \approx -\hat{\mu}_{j,2}$ the average will point in a random direction. This is avoided by defining μ_j as

$$\mu_j = \hat{\mu}_{j,1} + sign(\mu_{j,1} \cdot \hat{\mu}_{j,2})\hat{\mu}_{j,2} \qquad (5.13)$$

where

$$sign(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ -1 \text{ if } x < 0 \end{cases} \qquad (5.14)$$

On the twelve points on the polyhedron which have only five neighbours, μ_j has to be calculated differently. To get a symmetrical distribution of spheres, five spheres are needed, each sphere using the control point in which the tangent should be calculated, one neighbouring control point v_k and the two points neighbouring v_j but not v_k. The resulting normal is then calculated by generalizing (5.13):

$$\mu_j = \hat{\mu}_{j,1} + \sum_{k=2}^{N} sign(\hat{\mu}_{j,1} \cdot \hat{\mu}_{j,k})\hat{\mu}_{j,k} \qquad (5.15)$$

where N is the number of spheres (i.e 5).

Alternatively, only one sphere is calulated, hoping that three neighbouring points are enough for a good approximation.

Spheres approximating a mesh

Using 8-connectivity, the tangent plane of a mesh can be approximated with four spheres. Numbering the neighbours according to Figure 5.5 we calculate the four spheres using the following sets of points: {0, 4, 3, 8}, {0, 2, 6, 8}, {0, 5, 1, 6}, {0, 7, 1, 3}. The resulting $\mu_{j,1}$,..., $\mu_{j,4}$ are used according to (5.15) to calculate μ_j. The alternative mentioned above, i.e., to calculate only one of theses spheres, is still valid.
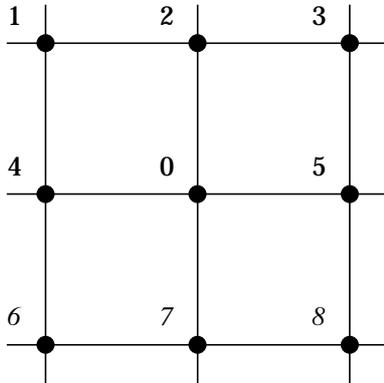

Figure 5.5: A numbering of control points.

Computational complexity

Using Cramer's rule [10] stating[1]

$$A^{-1} = \frac{adj(A)}{det(A)} \qquad (5.16)$$

to invert the matrix in (5.12), the inversion can be done making *27* floating point operations. (Since we do not care about the length of the vector μ_j, it is not necessary to calculate $det(A)$.) Creating the vector on the right hand in (5.12) needs *24* operations, and multiplying it with A^{-1} needs another *15* operations. This adds up to *66* operations to solve the system (5.12) and calculate one sphere.

Equation (5.15) needs *11·(N - 1)* operations plus another *9N* operations for normalizing the vectors. Summing up, there is need for *86N - 11* operations to calculate the normal vector.

Comparing this with the method suggested in Section 5.3 (using basis functions), sphere-fitting is faster except for on a very small mesh.

5.5 Controlling the Contour

Projecting the elasticity on the tangent plane before it influences the shape of the contour makes the contour more stable and less sensitive to the choice of parameters. Using a constant α of *0.5*, which makes the points move to the (currently) ideal position in each iteration, should no longer change the shape of the contour significantly.

By projecting the rigidity on the tangent plane as well, the contour can hold on to shapes in the images that have a high degree of curvature (without changing the parameter β in selected points). The contour has of course lost most of its rigidity but will now keep approximately the same size and shape (except for by influence from the attractor image).

The contour can now be controlled by adding another force, a user-controlled inflation force, to give the contour the preferred size. This makes it possible to "guide" the contour to the interesting part of the image by manipulating only one (global) parameter. This is the same idea as the balloon model described in [1].

For example, when trying to find the inside of a ventricle, a small spherical contour (polyhedron) could be placed in the middle of the ventricle and inflated to approximately the correct size. Then, the contour can be turned active again, by

---

1. $B = adj(A)$ denotes the matrix of which the element b_{ij} equals the cofactor of the element a_{ji} in A.

no longer projecting the rigidity and resetting the inflation force.

The force influencing a control point v_j can now be expressed as

$$F(v_j) = \alpha(F_{elast}(v_j) - (F_{elast}(v_j) \cdot \hat{\mu}_j)\hat{\mu}_j) + \quad (5.17)$$
$$\beta(F_{rigid}(v_j) - \varepsilon(F_{rigid}(v_j) \cdot \hat{\mu}_j)\hat{\mu}_j) +$$
$$\gamma\hat{\mu}_j - \nabla p(v_j, f)$$

where γ is the user-controlled parameter regulating the inflation force and ε is the parameter telling the contour to be active or inactive, i.e.,

$$\varepsilon = \begin{cases} 1 \text{ if } |\gamma| > 0 \\ 0 \text{ otherwise} \end{cases} \quad (5.18)$$

When calculating the inflation force it is important that all normal vectors μ_j are pointing in the "same" direction. That is, on a spherical contour, all normal vectors should point towards the outside of the contour (or inside, but they should all be alike). If the normal vectors are calculated using spheres, this is only guaranteed if the contour is convex. The problem is not trivial, but since it depends on the implementation it is not treated here. A solution is described in Appendix B, "Implementation in 3D".

EXPERIMENTS

*This chapter describes the experiments performed to test the meth-
ods suggested in earlier chapters. Two experiments were per-
formed; one with ideal, mathematically defined image volumes
and one with "real" medical image volumes.*

6.1 THE ACTIVE CONTOUR MODEL

The active contour model was implemented as
C++ classes and AVS modules, as described in
Appendix B.3, "The Active Contour Model".

The internal forces in the control points
$v_j \in \Re^3$ were computed as defined in Section 4.4,
i.e., as

$$F_{elast}(v_j) = c_{j,1} - v_j \qquad (6.1)$$

$$F_{rigid}(v_j) = 4c_{j,1} - c_{j,2} - 3v_j \qquad (6.2)$$

where $c_{j,i} \in \Re^3$ is the average of all control
points on the distance i from the control point v_j.
The iterations were performed according to

$$\bar{v}_{k+1} = \bar{v}_k + F(\bar{v}_k) \qquad (6.3)$$

where the force $F(v_j)$ in each control point is com-
puted as

$$\begin{aligned}
F(v_j) = \ &\alpha(F_{elast}(v_j) - \varepsilon_1(F_{elast}(v_j) \cdot \hat{\mu}_j)\hat{\mu}_j) + \quad (6.4) \\
&\beta(F_{rigid}(v_j) - \varepsilon_2(F_{rigid}(v_j) \cdot \hat{\mu}_j)\hat{\mu}_j) + \\
&\gamma\hat{\mu}_j + clip(\delta, \nabla p(v_j, f))
\end{aligned}$$

α, β and γ are the parameters regulating the elas-
ticity, the rigidity and the inflation force respec-
tively, and $\varepsilon_i \in \{0,1\}$ are the parameters telling
the internal forces to be active or inactive (i.e., if
they should be projected on the contour's tangent
plane or not). The function *clip* limits the image
force vector to the length δ.

Note that the iteration-method in Equation
(6.3) is not the same as Equation (2.9) (Euler's
method). Instead, each control point is moved
according to the sum of all forces. This method
converges somewhat slower but is extremely
easy to implement in *O(n)*-time.

The normal vectors were approximated using
the sphere-fitting method described in Section
5.4.

6.2 FITTING CONTOURS TO SYNTHETIC SHAPES

Two test volumes were constructed, one for test-
ing each representation (mesh and polyhedron;
see Chapter 4). The purpose to measure the per-
formance of the contours, since the intended
shapes are well defined. The volumes used were:

- The "half-sphere" volume for testing the poly-
  hedron representation: A $60 \times 40 \times 60$ pixel
  volume where the pixels on a half-sphere and
  the pixels in the circle that cuts the sphere
  have the value one. The rest of the pixels are
  set to zero.

  In addition, an area on the half-sphere has
  its values multiplied with three, thus
  intended to attract the contour more strongly
  and disturb the point distribution. The vol-
  ume is illustrated in Figure 6.1.
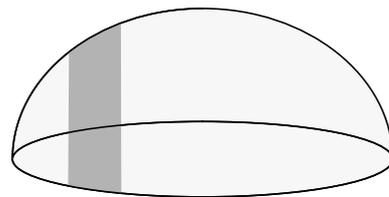
Figure 6.1: *The 'half-sphere'. The pixels
with higher value are darker.*

- The "half-cylinder" volume for testing the mesh representation: A $60 \times 80 \times 40$ pixel volume where the pixels on a half-circular cylinder are set to one and the other pixels are set to zero. Note that the cylinder is open in the ends. As in the "half-sphere" case, a band across the cylinder has higher values. The volume is illustrated in Figure 6.2.
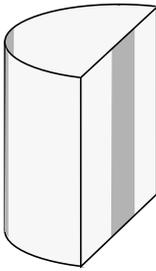
Figure 6.2: *The 'half-cylinder'. The pixels with higher value are darker.*

Creating attractor volumes

Both test volumes were filtered to create two different attractor volumes, henceforth called attractors, each:

1. Gradient. This attractor is defined according to the original edge detecting idea, i.e., $p(x, f) = |\nabla f(x)|^2$.

2. Orientation. The attractor is computed according to the steps in Section 3.5, but without any averaging of the tensor field. Since the test volumes lacks noise, the averaging is not necessary.

The attractors were computed using the GOP and AVS environments, as described in Appendix B.2, "Creating the Attractor Image".

The contour-fitting process

The following sequence of steps was followed when fitting a contour to a shape in a test volume. The sequence was performed twice for each attractor - once with the elasticity turned inactive and once without that possibility.

1. Place a small contour (with a diameter of about *10* pixels) in the middle of the test shape. A polyhedron with *42* control points was used in the "half-sphere" volume and a mesh with *56* control points was used in the "half-cylinder" volume.

2. Use *20* iterations for inflating the contour to approximately the right size and shape. The inflation force was set to one pixel, as was the maximum attractor force.

3. Expand the contour, i.e., increase the number of control points and halve the maximum attractor influence.

4. Iterate, and perform step three every fifth iteration.

Measuring the performance

To measure the success of the contour fitting, the contours were expanded twice more, after which the (Euclidian) distance from each control point to the test shape was computed. The average distance was used as a discrepancy measure.

The results from the polyhedron-case, using the 'half-sphere'-volume, are shown in the graph in Figure 6.4. As can be seen in the graph, the contour with active elasticity is closer to the intended shape after the initial *20* inflating iterations. This is because *20* iterations is too much (the contour grows too large), but the active elasticity reduces the size of the contour. When the inflation force is turned off, the contour with inactive elasticity performs better.
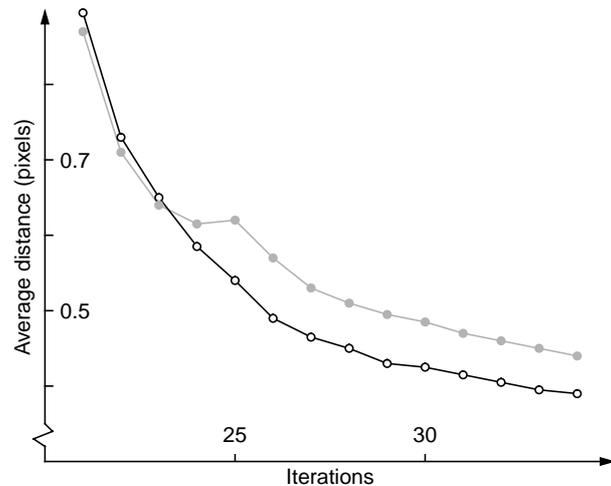
Figure 6.4: *Discrepancy measure of contours with inactive (black/white graph) and active (gray graph) elasticity. Both use the same orientation attractor.*

In the mesh case (using the "half-cylinder" volume), the contour with inactive elasticity had an average discrepancy of less than a third of a pixel, while the contour using active elasticity did not attach to the intended shape at all. Instead, an initial contour being larger than the intended shape had to be used, then letting the contour shrink under the influence of the elasticity. The final result was a discrepancy of only slightly less than one pixel.

Using a gradient attractor, the contour stabilized with a discrepancy of more than one pixel, which was expected since the contour attaches to the gradient attractor slightly beside the intended shape.

6.3 FITTING CONTOURS TO MEDICAL IMAGE VOLUMES

Two medical 3D image volumes have been used too - again, one for trying out each of the two representations. The purpose was to examine and give examples of how to use active contours in a more realistic situation, but without objective measurement of the performance (since the correct shapes are unknown). The volumes that were used are:

- The "knee" volume: This is a magnetic resonance-image of a knee of the size $256 \times 256 \times 128$ pixels. The bones of the knee and leg are clearly visible in the volume slices, and the intent was to fit a cylinder-shaped active contour to a part of one of the bones in the leg (either the thighbone or the calfbone).

  A 2D slice of the volume is shown in Figure 6.5, and it is obvious that an attractor image volume created with the gradient method could work well (the bone is surrounded by edges).

- The "heart" volume: This volume is constructed by a sequence of 2D ultra-sound images of a heart. The size is $177 \times 155 \times 152$, and different cavities (ventricles) can be seen by slicing the volume. It is very hard to determine the shape of the cavities by looking at the slices, and therefore the intent was to extract one of these shapes with an active contour represented by a polyhedron. A slice from the "heart" volume is shown in Figure 6.5, where a cross marks the cavity to which the contour was intended to fit.

### Creating the attractor volumes

Both medical volumes were filtered to create two different attractors each:

1. Gradient. The original edge-detecting method.

2. Averaged orientation. The attractor is computed according to the steps in Section 3.5 (including averaging).

The "knee" attractors

The main difference of the orientation attractors vs. the gradient attractor (see Figure 6.6, top two images) is that the orientation-attractors have their highest values slightly outside the surface of the bone. This is an effect of the fact that there are layers of tissue surrounding the bone, thus forming planar structure.

Another difference is that the orientation attractors are of considerable low-pass character when compared to the gradient attractor (which is no surprise - it was LP-filtered (averaged) while created). This makes it suitable to use the orientation attractor to find the first, rough approximation contour. Then we can increase the resolution of the contour and use the gradient
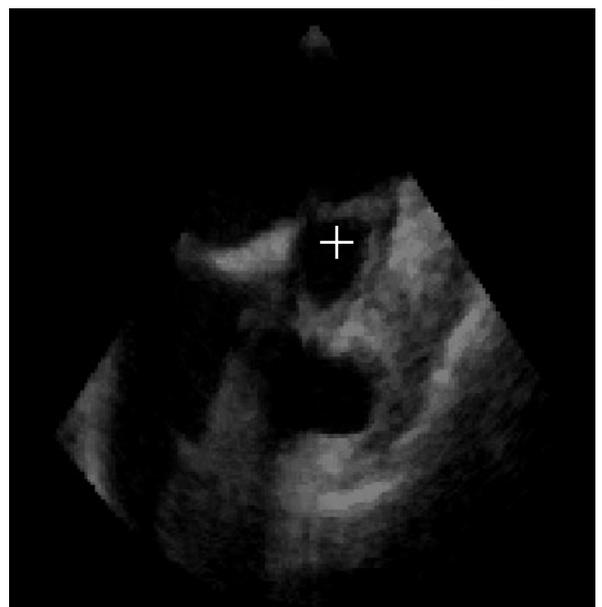
Figure 6.5: *A slice of the 'knee'-volume (left) and a slice of the 'heart'-volume (right).*

attractor to do a more exact fitting (of course, an averaged version of the gradient attractor could have been used for the first step).
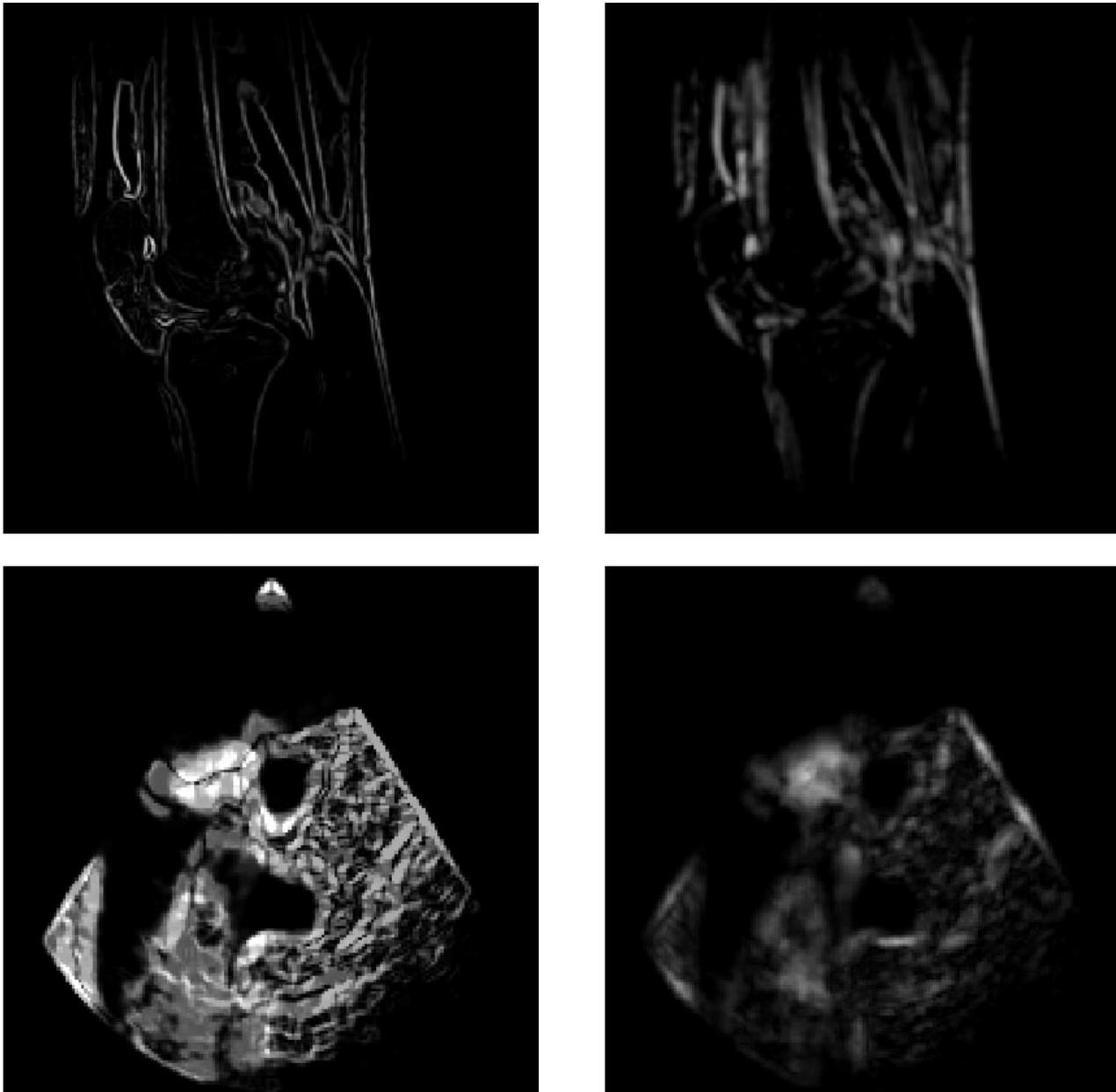
The "heart" attractors

The main difference between the types of attractors is that the gradient attractor has detected the surface enclosing the cavity to a very varying degree, leaving gaps and having some very strong points. Slices from two of the attractors are shown at the bottom of Figure 6.6.

The contour-fitting process

In both the "heart" and "knee" cases the sequence of steps followed was as follows:

1. Place an initial contour inside the sought-for surface. In the "knee" case, this was a plain cylinder with approximately half the diameter of the bone. In the "heart" case, it was an icosahedron.

2. Inflate the contour until it begins to be influenced by the attractor.

3. Increase the resolution of the contour.

Figure 6.6: *A slice of the gradient-attractor (top left) and the orientation-attractor (top right) of the 'knee'-volume. Below the corresponding attractors of the 'heart'-sequence.*

4. Do *10 - 20* iterations until the contour is quite stable.

5. If fine adjustment is needed, increase the resolution, decrease the maximum attractor influence and do some more iterations.

Fitting a contour to the "knee" volume

While fitting the contour in the 'knee'-volume the following observations were made:

- It is not necessary to let the control points be able to move along the cylinder-shaped contour - actually the contour "behaves better" if not. This is because if the control points can move along the cylinder, they tend to gather in the places where the attractor is especially strong, thus being inequidistant.

It is, however, important to compute the internal forces using all dimensions, thus keeping the resulting 2D contours strongly correlated.

Figure 6.7: *The surface of a bone visualized with an active contour. Two orthogonal slices of the image volume are shown, while the rest of the volume is invisible.*

- The most important step during the iterations is to fit the ends of the cylinder well to the volume. When these are fitted, they should be fixed and the rest of the contour iterated.

- Since the initial contour was not close to the intended shape, but a thin cylinder inside the bone, it was necessary to inflate the contour. It turned out to be extremely hard to do this in a controllable way without turning the elasticity inactive (i.e., project it on the contour's tangent plane).

- As suspected, the gradient attractor worked best for fine adjustment. The orientation attractors could also have been replaced by averaged gradient attractors, but they work well for "catching" the contour from the initial inflation.

The resulting contour is visualized along with two orthogonal slices of the "knee" volume in Figure 6.7. Studying several slices of the volume verifies that the contour follows the bone's surface, except where the contour's rigidity has forced the contour to be smoother.

### Fitting a contour to the "heart" volume

While fitting a polyhedron to the cavity in the "heart" volume, it was absolutely necessary to turn the elasticity inactive. The polyhedron representation needs a strong elasticity force to keep the control points fairly equidistant, and without turning the elasticity force inactive it will not let the attractor influence the contour enough.

With the "heart" volume, the averaged orientation attractor was the best to use. Since the gradient attractor varied greatly in strength, maximum possible elasticity was not enough to keep the control points from gathering in strong areas along the surface.

Using the orientation attractors, the problem still existed, but at a level that could be compensated by the elasticity force, and the contour finally fitted the cavity quite well. However, that is stated without any medical knowledge and without knowing the shape that such a cavity is supposed to have[1]. What was done was to compare the contour with several slices of the original volume.

The resulting contour is shown in Figure 6.8.
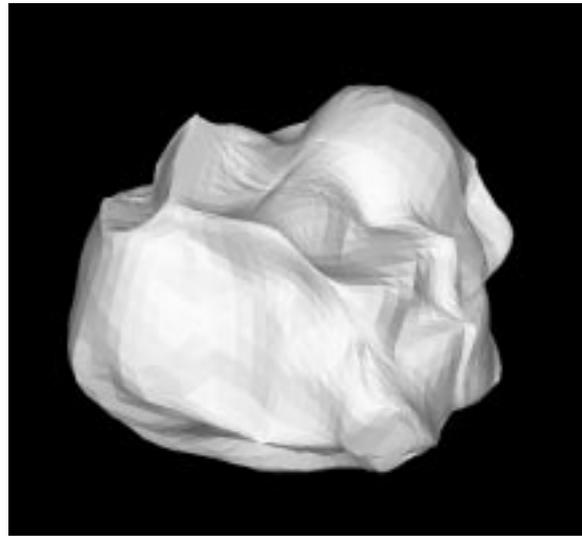
Figure 6.8: *The surface of a cavity in a heart visualized with an active contour.*

---

1. To add to the confusion, the ultra-sound volume is not sampled so that the angles are correct.

7

CONCLUSIONS & FUTURE WORK

This chapter discusses the results of the experiments and some ideas for future study of active contours.

7.1 THE ATTRACTOR IMAGE

Two ways of creating the attractor image were described in Chapter 3, "The Attractor Image"; the traditional edge-detecting gradient attractor and an orientation-based detector of planar structure.

The most suitable manner of creating the attractor image depends fully on the image the contour should be fitted to, and nothing general can be said. I have, however, shown how to use the orientation-based method for creating attractor images and that the method works well in certain situations.

There are, of course, an infinite number of possible ways to create attractor images. Depending on what feature that should attract the contour, any local feature extracting method could be used to create an attractor image.

Further studies

What should have been studied is the contours' sensitivity to noise using different attractors. Such experiments were not done, since other problems received higher priority.

In the "knee" volume (see Chapter 6, "Experiments") the gradient attractor worked well because it is variant to phase. It is, however, still a planar strucutre that is sought for, and an attractor sensitive to both phase and planar structure is probably the best solution in all the test volumes. This could be implemented by a one dimensional line/edge detecting in the direction of the eigenvector corresponding to the largest eigenvalue of the orientation tensor, or with a phase-variant orientation tensor (discussed in [11]).

7.2 THE REPRESENTATIONS

The two representations (mesh and polyhedron) described in Chapter 4, "Representation in Three Dimensions" worked well, as did the algorithms for expanding them and for calculating the internal forces[1].

Further studies

What should be further examined is how to expand the polyhedron exclusively in those areas where the control points are too widely spread. It will then be necessary to find ways to calculate the internal forces on a polyhedron with varying resolution.

The internal forces should also be redefined to be invariant to the resolution of the contour. With the methods described in Chapter 4, the spatial scope of the internal forces will be halved when the contour is expanded.

7.3 CONTROLLING THE CONTOUR

During the project, it turned out that it was very hard to control the contour when a strong elasticity was needed. The experiment with fitting an active contour to the cavity inside a heart seemed nearly impossible.

The solution described in Chapter 5, "Robust Control", i.e., to approximate the contour's tangent plane with sphere fitting and project the elasticity force on that plane (turning the elasticity inactive), solved the problem almost completely[2], and this is probably the most important result of this project. The experiments that were

---

1. The exact algorithms have not been described in the thesis, but can be found in Appendix B, "Implementation in 3D".

performed showed that turning the elasticity inactive produced better results as well as made the interactive process of fitting the contours easier for the operator.

Further studies

To make the contour fully usable in practical applications, a number of small controlling features should be developed, e.g., letting the contour be inflated in one direction, or fixing certain control points.

7.4 F*URTHER* S*UGGESTIONS*

Visualization

The main purpose of active contours in three dimensions is probably visualization, which was not treated in this thesis (the AVS environment did this work). However, some aspects could be studied further. For example, how should a contour passing empty spaces in the attractor image

---

2. For a complete solution, a polyhedron with adaptive resolution is needed.

be visualized? Suggestions are that it should either be flattened out, or that it should be transparent, depending on the application.

Complex models

A key to successful use of the active contour model is to have a good initial suggestion. Also, new internal forces could be added, keeping the contour from diverting too much from the suggested model.

It is also possible to define the internal forces differently in different parts of the contour. For example, when visualizing ventricles of the heart in time-sequences, some parts are known - in advance - to move more than others, and those parts should then be less rigid in the time-dimension.

Four dimensional contours

The ultra-sound volume used in Chapter 6, "Experiments" (the "heart" volume) is actually a volume sequence, i.e., a four dimensional image. An extension of the active contour concept to four dimensions could be useful, for example to measure the variation of the volume of a ventricle in time.

REFERENCES

[1] L. D. Cohen. On Active Contour Models and Balloons, *Computer Vision, Graphics and Image Processing: Image Understanding,* Vol 53 No 2, pages 211-218, 1991.

[2] L. D. Cohen and I. Cohen. A finite element method applied to new active contour models and 3D reconstruction from cross sections, *Proceedings of the International Conference on Computer Vision,* pages 587-591, 1990.

[3] I. Cohen, L. D. Cohen and N. Ayache. Using Deformable Surfaces to Segment 3-D Images and Infer Differential Structures, *Computer Vision, Graphics and Image Processing: Image Understanding,* Vol 56 No 2, pages 242-263, 1992.

[4] H. S. M. Coxeter. *Regular polytopes,* Dover Publications, 1973.

[5] L. Eldén and L. Wittmeyer-Koch. *Numerisk analys - en introduktion,* Studentlitteratur, 1987.

[6] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision,* Kluwer Academic Press, 1995.

[7] J. Hansson. *Segmentering och visualisering av kryoanatomidata,* Master's Thesis, Report LiTH-ISY-EX-1135, Linköping University, 1992.

[8] F. Herbert. *Chapter House Dune,* New English Library, 1985.

[9] M. Kass, A. Witkin and D. Terzopoulos. Snakes: Active Contour Models, *International Journal on Computer Vision,* Vol 1, pages 321-331, 1988.

[10] A. Migdalas and M. Göthe-Lundgren. *Kombinatorisk optimering - problem och algoritmer,* Dept. of Mathematics, Linköping University, 1994.

[11] K. Nordberg. *Signal Representation and Processing using Operator Groups,* PhD Thesis, Dissertation No. 366, Linköping University, 1994.

[12] G. Sparr. *Kontinuerliga system,* Dept. of Mathematics, Linköping University.

[13] C. J. Westelius. *AVS for beginners,* Internal report, Computer Vision Lab., Linköping University.

[14] G. Xu, E. Segawa and S. Tsuji. Robust Active Contours with insensitive parameters, *Pattern Recognition,* Vol 27 No 7, pages 879-884, 1994.

A

IMPLEMENTATION IN 2D

Early during the project, 2D active contours were implemented allowing basic experiments. This appendix describes the implementation in detail, so that the experiments are easily reconstructible. An example of the usage of this implementation can be found in Chapter 2.

A.1 COMMENTS ON IMPLEMENTING CONTOURS

- The control points should have continuous coordinates and so be able to be situated "between" pixels. This is necessary to get a stable solution, but it also makes it necessary to interpolate the attractor image.

- If no normalization of the image force is done, it may be necessary to set a maximum level on the influence from the attractor image.

A.2 AN IMPLEMENTATION IN MATLAB

The implementation is made in MATLAB 4.2c using the *Image Processing Toolbox* for showing (`imshow`) and filtering (`filter2`) images.

Complex coordinates are used, i.e., the coordinates *(x, y)* in the image are represented with the complex number $x + iy$. The reason for this is simplicity; the gradient vectors are represented by one complex number each, and the matrix indices become the same in the image and in the vector field.

Step 1: Suggesting the initial contour

To let the user suggest a contour, the image is displayed and the function `getsnake` is called:

```
>> load(blood)
>> imshow(blood)
>> v1 = getsnake('circ');
Suggest starting points with left
mouse button.
When finished, click right mouse button.
Calculating spline...
Placing points...
>> v2 = getsnake('circ');
```

```
Suggest starting points with left
mouse button.
When finished, click right mouse button.
Calculating spline...
Placing points...
```

The function `getsnake` reads the coordinates of mouse-clicks on the image, and calculates splines between the click points. Control points are then placed along the splines approximately every fourth pixel.

The argument to `getsnake` can be `'circ'` or `'line'` and determines whether the contour is to be open or closed. `getsnake` returns a vector of control points.

Step 2: Calculating the attractor image

Two gradient filters, g_x and g_y are used to create the attractor image.

$$g_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad g_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

By weighting these filters with unit length vectors (i.e., complex numbers) in the appropriate directions, one can create a gradient filter *g*:

```
>> g = gy + i*gx;
```

The attractor image, *p*, is then created with the function `absgrad2` which filters the image with the gradient filter in two scales. The results are added, normalized and squared as shown in Figure A.1:

```
>> p = absgrad2(blood,g);
```

33

Finally, the image force is calculated as a vector field:
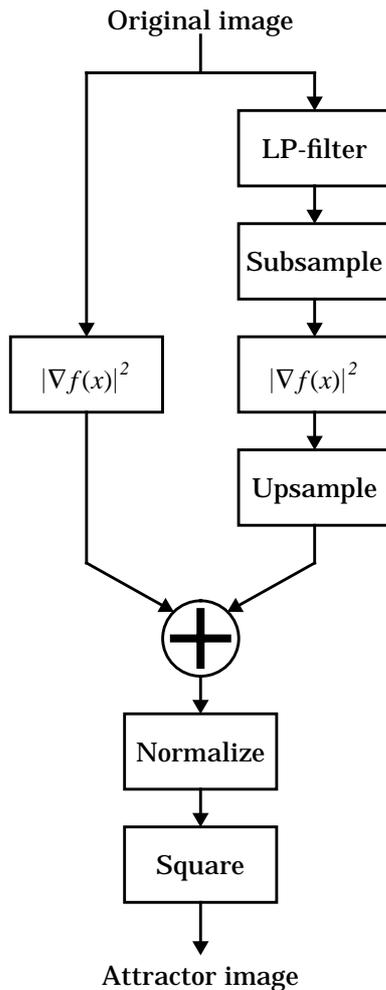
```
>> imgf = filter2(g,p);
```

Step 3: Iterate

The iteration is carried out by the function snake in the following way:

```
>> u1 = snake(0.25, 0.005, 0.5, imgf,...
   v1, 100, 'circ', 1, 4);
>> u2 = snake(0.25, 0.005, 0.5, imgf,...
   v2, 100, 'circ', 1, 4);
```

The parameters to this function are:

- w_elast and w_rgd are the α- and the β-function (constants) that determine the elasticity and rigidity according to (2.2). Larger values than the above given, *0.25* and *0.005* respectively, tend to give unstable contours.

- clip is the maximum allowed level of image force influence.

- imgf is the gradient of the attractor image to be used (calculated in Step 2 above).

- v is the originally suggested contour (a vector of control points) from Step 1 above.

- iterations determines the number of iterations to be done.

- snaketype is 'line' or 'circ' and determines if the contour is open or closed.

- mind and maxd determine the conditions for adding/removing control points. mind should be less than half of maxd.

The function snake uses the function snakemat, which calculates the matrix *A* according to (2.8), ip which makes the interpolations of the attractor image gradient and the functions mindist/maxdist which remove/add control points.
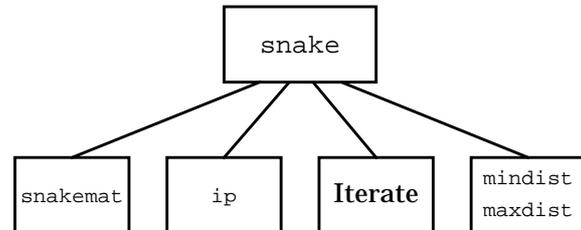
Figure A.2: *The* MATLAB *function* snake*.*

A.3 ESTIMATING LOCAL ORIENTATION

In Section 3.3, "Attracting to Local Orientation in 2D Images", the magnitude of local orientation is estimated in an image. This estimate is calculated using the MATLAB function absorient which works according to the same scheme as absgrad2 (see Figure A.1). However, the operation $|\nabla f(x)|^2$ is replaced with the function orientv that calculates the orientation vector of which the length equals the expression given in equation (3.7).

```
>> p = absorient(image,g1,g2,g3,g4);
```

The filters *g1,..., g4* given as arguments should be complex edge/line detectors, such as the following:

Figure   A.1:   *The   MATLAB-function* absgrad2*.*

$$g1 = \begin{bmatrix} 0 & 2 & 0 \\ -4 & 4 & -4 \\ 0 & 2 & 0 \end{bmatrix} + i \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$g2 = \begin{bmatrix} 2 & 0 & -4 \\ 0 & 4 & 0 \\ -4 & 0 & 2 \end{bmatrix} + i \cdot \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

$$g3 = \begin{bmatrix} 0 & -4 & 0 \\ 2 & 4 & 2 \\ 0 & -4 & 0 \end{bmatrix} + i \cdot \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$g4 = \begin{bmatrix} -4 & 0 & 2 \\ 0 & 4 & 0 \\ 2 & 0 & -4 \end{bmatrix} + i \cdot \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

A.4 MATLAB SOURCE CODE

The source code of all functions mentioned in this appendix follows in the order they are treated above.

Step 1: Suggesting the initial contour

MATLAB-function: getsnake

```
function u = getsnake(snaketype)

  disp('Suggest starting points with left
mouse button.');
  disp('When finished, click right mouse but-
ton.');
  t=1;
  while (t==1)
    [x,y,t] = ginput(1);
    u0 = [u0; x+i*y];
  end

  disp('Calculating spline...');
  clicks = length(u0);
  if snaketype == 'circ'
    u0(clicks) = u0(1);
  elseif snaketype == 'line'
    clicks = clicks-1;
    u0 = u0(1:clicks);
  end
  s = [0:clicks-1]'/(clicks-1);
  pp = spline(s,u0);

  disp('Placing points...');
  u = u0(1); n = 1;
  for j = 2:clicks
    dist = abs(u0(j) - u0(j-1));
    steps = round(dist);
    for k = 1:steps
      unext = ppval(pp, ...
                    (j-2+k/steps)/(clicks-1));
      if abs(u(n) - unext) > 3
        u = [u; unext];
        n = n+1;
      end
    end
  end
end
```

Step 2: Calculating the attractor image

MATLAB-function: absgrad2

```
function energy = absgrad2(img,g)

  energy = abs(filter2(g,img));

  img2 = subsample(img);
  energy2 = abs(filter2(g,img2));
  energy2 = upsample(size(img),energy2);

  energy = energy + energy2;

  energy = (energy/max(max(energy))).^2;

end
```

MATLAB-function: subsample

```
function img = subsample(img)

  sz = size(img);
  img = filter2(ones(3,3)/9,img);
  img = img(1:2:sz(1),1:2:sz(2));

end
```

MATLAB-function: upsample

```
function img = upsample(newsz,img);

  sz = size(img);
  x = newsz(1); y = newsz(2);
  ipx = (x-1)*(0:sz(1)-1)/(sz(1)-1);
  ipy = (y-1)*(0:sz(2)-1)/(sz(2)-1);
  img = interp2(ipx,ipy,img,...
  ones(x,1)*[0:x-1],...
  [0:y-1]'*ones(1,y));

end
```

Step 2: Iterate

MATLAB-function: snake

```
function v = snake(w_elast, w_rgd,...
                   clip_img, imgf,...
                   v,iterations,...
                   snaketype,mind,maxd)

  for j=1:iterations

    % Calculate the iteration matrix A
    if (maxd>0 | mind>0 | j==1)
      A = snakemat(w_elast,w_rgd,...
                   length(v),snaketype);
    end

    % Calculate grad(p(v,f)) by interpolation
    imgfv = ip(imgf,v);
    imgfv = min(clip_img*sign(imgfv),imgfv);
    if(snaketype=='line')
      imgfv(1)=0; imgfv(length(imgfv))=0;
     end

    % The iteration step
    v = v + imgfv - A*v;

    % Add/remove control points
    if mind>0; v = mindist(mind,v); end
    if maxd>0; v = maxdist(maxd,v); end

  end
```

MATLAB-function: snakemat

```
function A = snakemat(elast,rgd,n,snaketype)

  B = [elast -rgd]*[0 -1 2 -1 0; 1 -4 6 -4 1];
  B = ones(n,1)*B; d = -2:2;
  if snaketype == 'circ'
    B = [B(:,5:-1:4) B B(:,2:-1:1)];
    d = [2-n 1-n -2:2 n-1 n-2];
  elseif ~(snaketype == 'line')
    error('Use snaketype "line" or "circ"');
  end

  A = spdiags(B,d,spalloc(n,n,5*n)...
                  +sparse(1,1,1,n,n));

  if snaketype == 'line'
    A(1,:) = zeros(1,n);
    A(n,:) = zeros(1,n);
    A(2,1:4) = elast*[-1 2 -1 0];
    A(n-1,n-3:n) = elast*[0 -1 2 -1];
  end

end
```

MATLAB-function: ip

```
function y = ip(img,coords)

  sz = size(img);
  y = interp2(1:sz(1),1:sz(2),img,...
              real(coords),imag(coords));

end
```

MATLAB-function: mindist

```
function u = mindist(dist,u);

  for j=2:length(u)-1
    if(abs(u(j)-u(j-1)) < dist...
       & abs(u(j)-u(j+1)) < dist)
      u = remove(j,u);
      u = mindist(dist,u);
      break;
    end
  end

end
```

MATLAB-function: maxdist

```
function u = maxdist(dist,u);

  L = length(u);
  for j = 1:L-1
    vector = u(j+1) - u(j);
    q = abs(vector)/dist;
    if q > 1
      newpoints = floor(q);
      newvector = vector/(newpoints+1);
      for k=1:newpoints
        u = insert(u(j)+k*newvector,j+k,u);
      end
      u = maxdist(dist,u);
      break;
    end
  end

end
```

Estimating local orientation

MATLAB-function: absorient

```
function grad = absorient(img,g1,g2,g3,g4)

  grad = abs(orientv(img,g1,g2,g3,g4));

  img2 = subsample(img);
  grad2 = abs(orientv(img2,g1,g2,g3,g4));
  grad2 = upsample(size(img),grad2);
  grad = grad + grad2;

  grad = (grad/max(max(grad))).^2;

end
```

MATLAB-function: orientv

```
function or = orientv(img,g1,g2,g3,g4)

  Q1 = abs(filter2(g1,img));
  Q2 = abs(filter2(g2,img));
  Q3 = abs(filter2(g3,img));
  Q4 = abs(filter2(g4,img));

  or = Q1-Q3 + i*(Q2-Q4);

end
```

B

IMPLEMENTATION IN 3D

This appendix is an overview of the software that was developed while implementing 3D active contours. The class hierarchy is explained, as well as some non-trivial algorithms for polyhedrons.

B.1 ENVIRONMENT

Some notes on the environments used are included to make the rest of the appendix understandable.

Image catalogues

3D image volumes are stored in *image catalogues*. An image catalogue is a directory which contains several files; one for each dimension of the elements in the image for each volume slice. The gradient field of an image with the size $10 \times 10 \times 10$ will therefore be stored as *30* files. This is, however, quite transparent to the user.

GOP

The GOP, *General Operator Processor*, is a specialized computer for image processing operations. The operations are started from a UNIX terminal with the command `run` followed by more specific orders. The GOP performs its operations on image catalogues.

AVS

AVS, *Application Visualization System* from *Advanced Visual Systems, Inc.*, is a modular system that lets the user build networks connecting modules each performing a specific task. The modules can be predefined, or the user can write his own modules in C or Fortran. There is a large amount of predefined modules available for filtering images, presenting data, performing mathematical operations, etc.

An example of a very simple AVS network is shown in Figure B.1, where the predefined mod-
ules `animated integer`, `field math` and `field to byte` are used with the module `AttractorImage` (written as a part of this project) and the modules `PickICimage` and `eigenvectors` provided by the Computer Vision Lab.

Step two of the segmenting process (see Section 2.3, "The Contour-fitting Process"), i.e., calculating the attractor image, is performed partly by an AVS network. The third step, the iteration, is performed fully by an AVS module (`Active-Contour`).

For information on the AVS data structures, see the AVS Developer's Guide or [13].

B.2 CREATING THE ATTRACTOR IMAGE

The attractor image is created according to the six steps described in Section 3.5, "Creating the Attractor Image".

Step 1: Tensor filtering

This step is performed with the GOP:

```
run trace in=original out=tensor, s16, s16,
s16, s16, s16, s16 krn=quad3D7a2b
```

The input is an image catalogue, `original.ic`, containing the original image. The output is a new image catalogue, `tensor.ic`, where each pixel has a six-dimensional value.

Step 2: Averaging

This step is performed with the GOP:

```
run saver3D in=tensor out=tensor_aver, s16,
s16, s16, s16, s16, s16 S16=1
```

37

Step 3: Eigenvalues

The tensor-filtered image catalogue is loaded into the AVS environment and the module `eigenvectors` is used to calculate the eigenvalues. The AVS network used is shown in Figure B.1. The module `animated integer` makes the module `PickICimage` traverse an image catalogue and send each image to `eigenvectors`, which passes the vector field of eigenvalues on to `AttractorImage`.

Steps 4 and 5: Finding planar structure and thresholding

The estimate of planar structure, as well as the thresholding, is made by the AVS-module `AttractorImage` in the network shown in Figure B.1. Depending on the parameter "`Delta`" (a toggle button) the user can choose which of the two suggested certainty estimates is used.

The module contains the function `Compute` which is called each time the value on the input port is changed. This function does its computation on one slice of the image and puts the result on the output port.

Step 6: Gradient filtering

The gradient filtering is most easily performed by the GOP:

```
run grad3D in=attractorimage out=imageforce,
s16, s16, s16
```

B.3 THE ACTIVE CONTOUR MODEL

The active contour model is implemented as an AVS module operating on classes representing contours and vector fields. The module is suitably included in a network as the one shown in Figure
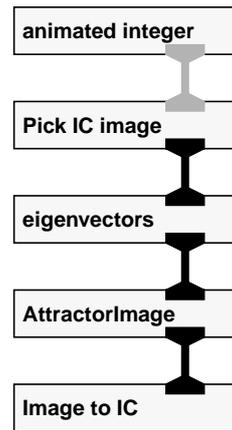

Figure B.1: *The AttractorImage network.*

B.2. The "Read IC" modules read image catalogues into 3D fields - the left hand module reads the image force field, and the right hand module reads the original image for visualization. The output (through the geometry viewer) is an image such as the one shown in Figure 6.7.

The AVS module ActiveContour

The module `ActiveContour` can be compiled into two different modules, `ActiveTube` and `ActiveSphere`, using the mesh and the polyhedron representation of an active contour respectively.

The module has one input port, the image force as a 3D field of 3D vectors, and two output ports. The output ports both deliver the current shape of the active contour, one as a field of 3D-vectors and the other as an AVS geometry.

The module's parameters are controlled through the user-interface shown in Figure B.3.
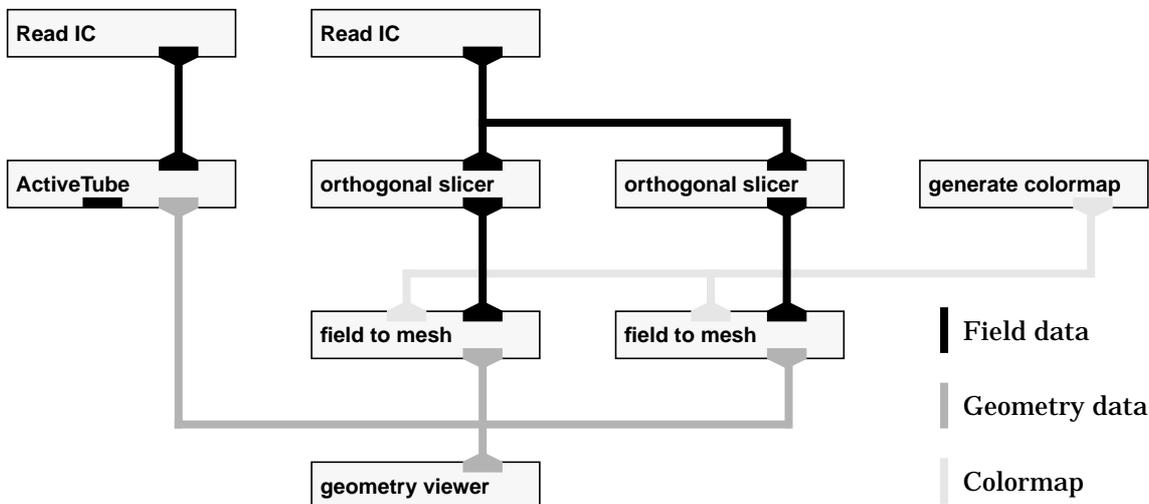

Figure B.2: *The ActiveContour network.*

Go one makes the contour iterate once.

Go many starts a chain of iterations.

Iterations tells how many iterations should be made when pressing "Go many".

Expand increases the number of control points on the contour.

Clip Image Force determines the maximum influence from the image (measured in pixels).

Active elasticity (rigidity) determines if the elasticity (rigidity) force should be projected on the contour's tangent, i.e the parameter ε_1 (ε_2) in equation (6.4).

Elasticity is the (normalized) strength of the elasticity, i.e the parameter α in equation (6.4).

Rigidity is the (normalized) strength of the rigidity force, i.e the parameter β in equation (6.4).

Inflation is the strength of the inflation force (in pixels), i.e the parameter γ in equation (6.4).

Fixed ends makes the ends of the cylinder-shaped contour be fixed (only applicable on a contour represented by a mesh).

Fixed Y makes the control points immovable in the *y*-dimension (only applicable on a contour represented by a mesh).

Load loads a contour from a file on disk into the module.

Initial contour is a filebrowser to choose the file to load.

Save saves the current contour to a file on disk.

Resulting contour is a filebrowser to choose the file to load.

Figure B.3: *The user-interface of the ActiveContour module.*

Overview of the class hierarchy

The active contour model is implemented using six C++ classes. The relationships between these classes are illustrated in Figure B.4.

The "main" classes are the two contour classes, `Polyhedron` and `Mesh`, which are used by the `ActiveContour` module for representing the active contour, and the class `Field`, which is used for representing the vector fields such as the image force, the elasticity force, etc.

The purpose and function of the classes are as follows:

- Class `Field`

  This class is a basic storage class for 2D fields of 3D vectors. It is used for storing the computed values of the forces that influence the contour, as well as for storing the data of the contour itself. In addition, the class contains methods for interpolation of vector fields (since the image force field needs to be inter-

Figure B.4: The class hierarchy.

polated) and for projection of a vector field on another (since the elasticity should be projected on the contour's tangent plane).

- Class `Contour`

  This is a representation-independent abstract class for active contours providing AVS compatibility. The class contains methods for converting the contour to an AVS geometry or an AVS field, and also methods for fitting spheres to sets of vectors (used for approximating tangent planes according to equation (5.12)).

  The class `Contour` declares methods for creating polygons (for visualization), expanding ("upsampling") the contour) and for determining which control points to use for approximating the tangent planes (using the sphere-fitting method mentioned above). Since these methods are dependent on the representation of the contour (mesh or polyhedron), they are only declared, not defined, thus making `Contour` an abstract class.

  The class `Contour` is a subclass of the class `Field`, from which all storage handling is inherited.

- Class `Mesh`

  The class `Mesh` is a subclass of the class `Contour`, i.e., it is the mesh representation of an active contour. Representation-independent methods are inherited from the class `Contour`, and the representation-dependent methods (declared in the class `Contour`) are defined in this class.

- Class `Polyhedron`

  The class `Polyhedron` is the polyhedron representation of an active contour and is a subclass of the class `Contour` (thus inheriting representation-independent methods for active contours).

  The class `Polyhedron` contains methods for keeping track of how the vertices are connected (using the class `Connection`). In addition, the class defines the representation-dependent methods declared in the class `Contour`.

- Class `Connection`

  This is a primitive class keeping track of how the vertices in a polyhedron are connected.

- Class `Vector`

  The class `Vector` is used by all other classes except `Connection`. It is a simple class for 3D vector arithmetics.
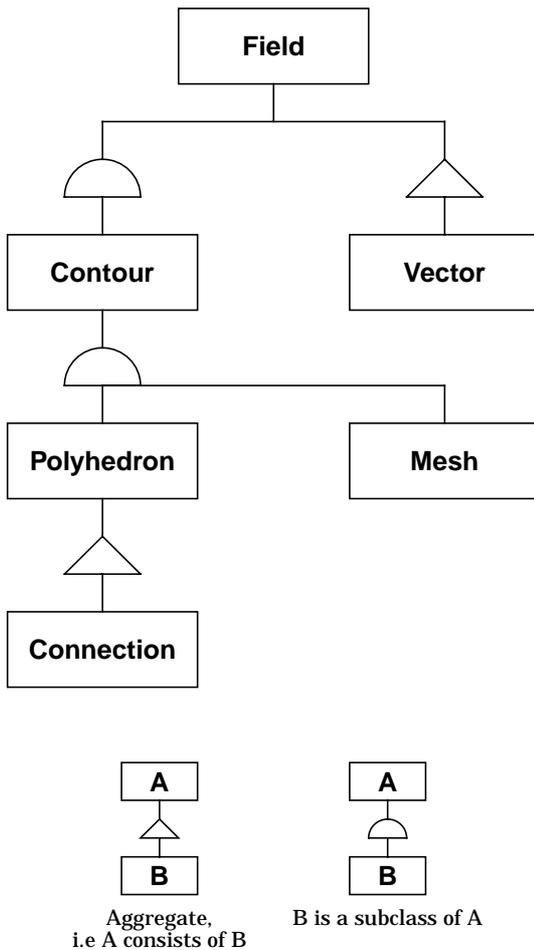
Compatibility with other visualization packages

The subclasses of class `Contour` (`Mesh` and `Polyhedron`) have built-in methods for converting themselves to an AVS field or an AVS geometry, and all classes can write/read themselves to/from a stream in ASCII-format. There should be no problems adding IO-methods for compatibility with any other visualization package by adding the necessary methods to the classes `Mesh` and `Polyhedron`.

Polyhedron expansion and connection ordering

For visualization purposes the connection data for each vertex in the class `Polyhedron` is ordered counter-clockwise, i.e., if looking at the (visualization of the) polyhedron, the connection list corresponding to a vertex (control point) v_j should point out vertices surrounding v_j in counter-clockwise order.

To keep this order when expanding the polyhedron, the following algorithm is used when expanding:

1. Expand the polyhedron by placing a new vertex at the middle of each edge, i.e., on the average between two connected vertices.

2. For each old vertex (v_a), replace each connection to another vertex (v_b) with a connection to the new vertex being the average of v_a and v_b.

3. For each old vertex, traverse the connection list (now containing only new vertices). If two vertices in the (cyclic) connection list are neighbours then connect them to each other.

4. For each new vertex, sort the connections so that the connections are neighbours in the (cyclic) connection-list if and only if the corresponding vertices are connected.

5. For each new vertex v_i, call the two first vertices in its connection list v_k and v_l. If v_i precedes v_l in v_k's connection list, then reverse v_i's connection list.

The reader is invited to verify the validity of the algorithm as an exercise of geometry.

Second order connections

When computing the rigidity force in a polyhedron, the average of all vertices on a distance of two steps from each vertex is needed. Therefore, the class `Polyhedron` for each vertex also stores a "second order" connection list, keeping track of all vertices two steps away. This connection list is created when expanding the polyhedron in the following way: For each vertex v_j, for each vertex v_k connected to v_j, for each vertex v_l connected to v_k, if v_j and v_l are not connected and not equal, then create a second order connection between them.

Titel Title	Aktiva konturer i tre dimensioner Active Contours in Three Dimensions
Författare Author	Jörgen Ahlberg

Sammanfattning
Abstract

To find a shape in an image, a technique called *snakes* or *active contours* can be used. An active contour is a line that moves towards the sought-for shape in a way controlled by internal forces - such as rigidity and elasticity - and an image force. The image force should attract the contour to certain features, such as edges, in the image. This is done by creating an attractor image, which defines how strongly each point in the image should attract the contour.

In this thesis the extension to contours (surfaces) in three dimensional images is studied. Methods of representation of the contour and computation of the internal forces are treated.

Also, a new way of creating the attractor image using the orientation tensor to detect planar structure in 3D images is studied. The new method is not generally superior to those already existing, but still has its uses in specific applications.

During the project, it turned out that the main problem of active contours in 3D images was instability due to strong internal forces overriding the influence of the attractor image. The problem was solved satisfactory by projecting the elasticity force on the contour's tangent plane, which was approximated efficiently using sphere-fitting.

95-11-01/lli