Robust Fitting of Implicit Polynomials with Application to Contour Coding

Amir Helzer

This page intentionally left blank

Robust Fitting of Implicit Polynomials with Application to Contour Coding

Research Thesis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Electrical Engineering

By
Amir Helzer

Submitted to the Senate of the Technion
Israel Institute of Technology
AV 5760 Haifa June 2000

This page intentionally left blank

The research thesis was done under the supervision of Prof. David Malah and Dr. Meir Barzohar in the faculty of electrical engineering.

I would like to thank them both for their knowledgeable guidance.

This work was made possible thanks to the support of my family.

First, I thank my wife, Orit and her parents for their active support and encouragement.

I thank my mother, Drora, my brother, Yoav, and my grandparents, Baruch and Haviva.

The generous financial support of the Technion is gratefully acknowledged.

Dedicated to Tomer

This page intentionally left blank

Contents

List of figures

List of tables

This page intentionally left blank

Abstract

This work deals with the fitting of 2D and 3D implicit polynomials (IPs) to 2D curves and 3D surfaces, respectively, and its application to contour coding. The zero-set of the polynomial describes the data, and is determined by the IP coefficients.

IPs are currently being used for object recognition. The existence of geometric invariants to affine transformations have made them particularly attractive for recognition of objects that have undergone an unknown affine or rotational transformation. In this work we investigate the effectiveness of using IPs for contour coding. The motivation for using IPs for coding lies in their high description power, where a small number of polynomial coefficients may describe complex 2D curves or 3D surfaces.

In order to efficiently describe object contours using IPs, their coefficients must be quantized. This quantization effects the zero-set and leads to representation errors. A major topic in this work is the development of a fitting algorithm that produces IPs which are robust to quantization noise. We begin with an analysis of the zero-set errors caused by coefficient errors (such as quantization). Using the results of this analysis we examine the error properties of IPs obtained through the use of existing fitting algorithms. We derive our own IP fitting algorithm, which is aimed at minimizing the zero-set errors resulting from coefficient quantization, and compare its performance to existing algorithms. Our simulations support the analysis, which claims that the proposed fitting algorithm out-performs existing fitting algorithms in terms of robustness to quantization. We also show that using this algorithm improves the fitting even when the polynomial coefficients are unquantized.

In order to offer a complete coding scheme based on IPs, two more issues need to be addressed: curve segmentation and data reconstruction.

We show that complicated curves have to be segmented into several simpler curve segments in order to achieve low rate and reduced distortion. The segmentation of curves into several efficiently coded curve segments proves to be a complex problem, resulting from the high complexity of an exhaustive search needed for an optimal segmentation and the large space of possible solutions. We propose two sub-optimal solutions to the problem, being, top-to-bottom and bottom-to-top curve segmentation. Due to the sub-optimality, each solution has its advantages and disadvantages. From a comparison of the properties of these algorithms and from a comparison of the

segmentation results, we conclude that the bottom-to-top curve segmentation algorithm, as presented, is more efficient and produces better results than the top-to-bottom algorithm. This algorithm begins with minimal size segments, described by 1st order IPs. Segments are merged until a distortion limit is reached, and than the order of the polynomials is incremented. The merge/increment process continues until the maximal allowed IP order is reached. At the end of this process a scan of all the segmentations encountered is made and the segmentation with the lowest rate is selected.

Having provided solutions to both curve segmentation and polynomial fitting, we address the problem of data recovery from polynomial coefficients. We introduce geometric constraints on the polynomial zero-set, which are added to the fitting algorithm. These constraints guaranty that the part of the zero-set of the polynomial, which represents the data, is separated from any possible spurious zero-set points. Using these constraints, along with a starting point for the zero-set scan, allows restorability of the coded curve from the IP coefficients. We also describe a method for merging zero-set points from several IPs, representing the data of neighboring curve segments.

Finally, we describe a complete coding scheme, based on the above topics: curve segmentation, polynomial fitting and data reconstruction. The rate and distortion results of object contours coded by this scheme are compared to chain-coding. From this comparison we conclude that IP based contour coding is effective and can achieve good compression, depending on the image. It is also clear that the high computational complexity involved in this type of coding, restricts its use to applications where contour data makes up most of the image information, and the high complexity is justified.

We also suggest directions for further study. Some of these directions are improvements to algorithms presented in this work, and some are applications of the development done here to other problems. Specially appealing could be the applications of 3D IPs to image sequences, where it may be possible to use a single IP to describe the contour of an object found in several frames, even when the contour moves or changes.

Chapter 1 Introduction

This work deals with the problem of fitting implicit polynomials to 2D contours [1] and its application to contour coding.

Implicit polynomials (IPs) have applications in the field of object recognition and computer graphics. The purpose of this work is to present improved fitting of IPs to object boundaries and investigate the possible application of IP to contour coding. For the typical application of IPs, i.e. object recognition, tight and noise-immune fitting is required. This requirement is necessary for contour coding as well, but other requirements are unique to coding applications. This work proposes solutions to the requirements for both applications – object recognition and coding.

Whereas for object recognition the coefficients of the polynomials are used as features to compare between different objects; for coding applications, the goal is to use as few bits as possible to describe an object. Rate-distortion criteria which accompanies coding, motivates the derivation of a robust fitting and curve segmentation algorithms, both aiming at reducing the distortion and the number of bits required for coding contours.

The presentation of the various topics in this work begins with the general case of polynomial fitting to 2D curves, and continues with other topics, which are more specific for contour coding.

We noticed that in order to describe complicated curves (such as object boundaries) using IP segmentation of these curves into smaller and simpler curve segments is essential. This allows description of complicated object curves by several low order, efficient, and stable polynomials. This segmentation is especially useful for coding applications where low bit rate and low distortion is the goal. Using several low order polynomials (1^{st} to 4^{th} order), instead of a single high order polynomial, that can be fitted to the entire boundary (up to 18^{th} order), allows representation of the boundary using IPs with a low overall rate and distortion.

We also addressed the restoration problem of data represented by implicit polynomials. A problem which especially arises when fitting high-degree implicit polynomials to 2D and 3D data sets, is that the zero-sets may consist of multiple components, may be unbounded, or may fit the data in very irregular ways. This

problem is successfully solved in [19] by restricting the polynomial into parameterized families of polynomials with topological properties. We have chosen to implement geometric restrictions on the polynomial that shape the zero set so the original data can be restored from it.

Special emphasis was placed on the stability and restorability of the contour data from the quantized polynomial coefficients. While earlier work [2,3] which produced polynomials describing contour data was mainly aimed at recognition, we are interested in compression. Therefore, stability is of major interest to our polynomial fitting algorithm. A stable solution (low sensitivity of the location of the zero-set to errors in the polynomial coefficients), introduced in this work, produces polynomials that can be coded with fewer bits and are therefore more efficient for coding applications. As a consequence of the sensitivity analysis, on which the fitting algorithm was based, we show that this algorithm is also preferable for recognition applications (due to improved fitting results even without coefficient quantization).

The fitting algorithm can be used for coding applications once it is implemented with geometric constraints, also presented in this work. We state the geometric constraints that prevent spurious zero-set points from being restored as data.

In another part of the work – contour segmentation – we investigate different ways for segmenting contours into segments that are efficiently coded. The segmentation is necessary because unsegmented contours may lead to polynomials of very high order, requiring a large number of bits for a stable representation.

In the context of contour segmentation we also address the problem of data redundancy which arises due to overlapping contours in neighboring bodies. We explore two options for coding the contours – either as closed contours (around each body) or as shared curves (between each two neighboring bodies).

Although, in the coding scheme, curve segmentation precedes polynomial fitting, the latter is a basis for the segmentation process and is therefor described first.

Contour segmentation results in segmented contours, which need to be coded. We use implicit polynomials do describe these contours, where the zero set of the polynomials represents the contour data and the polynomial coefficients, combined with some necessary side information, comprise the code.

The outline of this work is as follows:

Chapter 2 provides background information on topics presented in the work. Chapter 3 describes the robust fitting algorithm, developed in this work, along with the sensitivity analysis on which it is based. Simulations results comparing quantization errors and unquantized fitting are also presented in this chapter.

Chapter 4 presents two curve segmentation algorithms, which aim to minimize the rate and distortion associated with the coded curve segments. Two curve segmentation algorithms are presented and compared – top-to-bottom and bottom-to-top algorithms.

Chapter 5 deals with data recovery and describes constraints placed on the polynomial fitting which enable data reconstruction. This chapter also presents a curve-merging scheme that is used to reconstruct the fitted data.

Chapter 6 describes the integration of the algorithms described in previous chapters into a complete contour coding algorithm. In this context simulation results of coding the contour information of segmented images are presented. Distortion and rate properties are shown, with compression results compared to those obtained by chain coding.

Chapter 7 summarizes the work, draws conclusions and suggests research directions for future study.

This page intentionally left blank

Chapter 2 Background

In this chapter we provide the background on the topics that this work is dealing with. Since the main topic of the work is polynomial fitting, this subject is presented in detail. We state the problem of polynomial fitting and present two fitting algorithms, which stand as milestones.

Next, we give some background material on contour coding, including the popular chain coding.

2.1 Fitting 2D implicit polynomials to curves

The object of polynomial fitting is to describe data points (object boundary for 2D objects or surfaces for 3D objects) with the zero-set of a polynomial. Therefore, we would like the value of the polynomial to be zero at the location of the data points.

The value of the polynomial at a point (x,y) can be described as the product of two vectors – a parameter vector (containing the polynomial's coefficients), and a vector of monomials.

For a d^{th} degree polynomial, the monomial vector is denoted as:

$$\begin{aligned}&\bar{p}(x,y)=[p_1(x,y),...,p_r(x,y)]=\\&\left[x^0y^0,x^1y^0,x^0y^1,...,x^dy^0,x^{d-1}y^1,...,x^1y^{d-1},x^0y^d\right]\end{aligned} \tag{2.1}$$

where $r=(d+1)(d+2)/2$ and the parameter vector is $\bar{a}=[a_1,a_2,...,a_r]$.

The value of the polynomial described by $\bar{a}$ at location (x,y) is:

$$P_{\bar{a}}(x,y)=\bar{a}\,\bar{p}^T(x,y) \tag{2.2}$$

We are looking for a parameter vector $\bar{a}$ that leads to a polynomial that best fits the data under a criterion to be specified. The data set contains N points with coordinates (x_n,y_n), $n=1,...,N$.

We denote the zero-set of the polynomial defined by the coefficient vector $\bar{a}$ as:

$$Z_{\bar{a}}=\{(x,y):P_{\bar{a}}(x,y)=0\} \tag{2.3}$$

2.2 Overview of existing fitting algorithms

We now give an overview of two polynomial fitting algorithms. The first algorithm presented here was originally presented by Taubin [4] and later improved [5,6]. The second algorithm – *3L* – was developed by Lei, Blane, and Cooper [7,8,9,10].

Today, it is known that Taubin's fitting algorithm is much less stable than the *3L* algorithm (this property is also proved in section 3.3). However, for the purpose of providing a complete description of the evolution of polynomial fitting, this algorithm is described here.

2.2.1 Overview of Taubin's non-linear algorithm

This algorithm seeks to minimize the sum of squared distances between each data point (x_n, y_n) and the closest point on the polynomial zero-set.

An approximation for the sum of squared errors is given by (the development of this approximation is given in Appendix A):

$$E \approx \sum_{n=1}^{N} \left(\frac{P_{\bar{a}}(x_n, y_n)}{\left\| \nabla P_{\bar{a}}(x_n, y_n) \right\|} \right)^2 \tag{2.4}$$

Taubin developed two methods for minimizing this expression. The method we used in our simulations is described here. The other method, which we found to be more sensitive numerically, is described in Appendix A.

Taubin suggested an iterative solution to this minimization problem, where the argument is evaluated in two parts. We refer to the denominator in (2.4) as relative weights for the different error components and rewrite the squared error as:

$$\sum_{n=1}^{N} \left(\frac{P_{\bar{a}}(x_n, y_n)}{\left\| \nabla P_{\bar{a}}(x_n, y_n) \right\|} \right)^2 = \sum_{n=1}^{N} \left(\frac{P_{\bar{a}}(x_n, y_n)}{w_n} \right)^2 \tag{2.5}$$

First, the denominator is calculated $w_n = \left\| \nabla P_{\bar{a}}(x_n, y_n) \right\|$ and second, a coefficient vector that minimizes the numerator is found. As an initialization, all denominator components are initialized to one - $w_n(0) = 1$.

Representation as a quadratic term enables a minimization of the numerator. For iteration j, the coefficient vector is calculated based on the weights from iteration $j-1$ by:

$$E(j)=\sum_{n=1}^{N}\left(\frac{P_{\bar{a}(j)}(x_n,y_n)}{w_n(j-1)}\right)^2=\sum_{n=1}^{N}\left(\frac{\bar{a}(j)\bar{p}^T(x_n,y_n)}{w_n(j-1)}\right)^2= \\ \bar{a}(j)\left(\sum_{n=1}^{N}\frac{\bar{p}^T(x_n,y_n)\bar{p}(x_n,y_n)}{w_n^2(j-1)}\right)\bar{a}(j)^T=\bar{a}(j)SM\,\bar{a}(j)^T \tag{2.6}$$

This expression is minimized when the coefficient vector - $\bar{a}(j)$ is selected as the eigenvector with the lowest corresponding eigenvalue (λ_{MIN}) of the matrix SM (referred to as a 'scattering matrix'). It should also be noted that since all eigenvectors have a unit size, the selected solution is never the trivial solution ($\bar{a}(j)=0$).

The result of this selection yields an error $E=\lambda_{MIN}{}^2$.

No proof of convergence of this algorithm was found in the literature. Experiments show, however, fast convergence (typically within 5 iterations) of the algorithm to an acceptable fitting result.

The latter (iterative) solution proved more stable and was used in our simulations.

2.2.2 Overview of the 3L algorithm

The *3L* linear solution developed by Lei et. al. [7,8,9] contains a criterion for fitting an implicit polynomial to a data set.

The algorithm proposes the construction of two additional data sets, which are based on the original data set. The two additional data sets are constructed so that one set is internal and the other is external relative to the original data set, with a distance d from it (see Fig. 2.1). The goal of the algorithm is to find a polynomial that has a value of zero for points on the original data points, ε on the internal added points and $(-\varepsilon)$ on the external added points.

To achieve this goal, the *3L* algorithm uses a *least squares* solution that minimizes the sum of squared errors between the required and actual polynomial values at the three data sets.

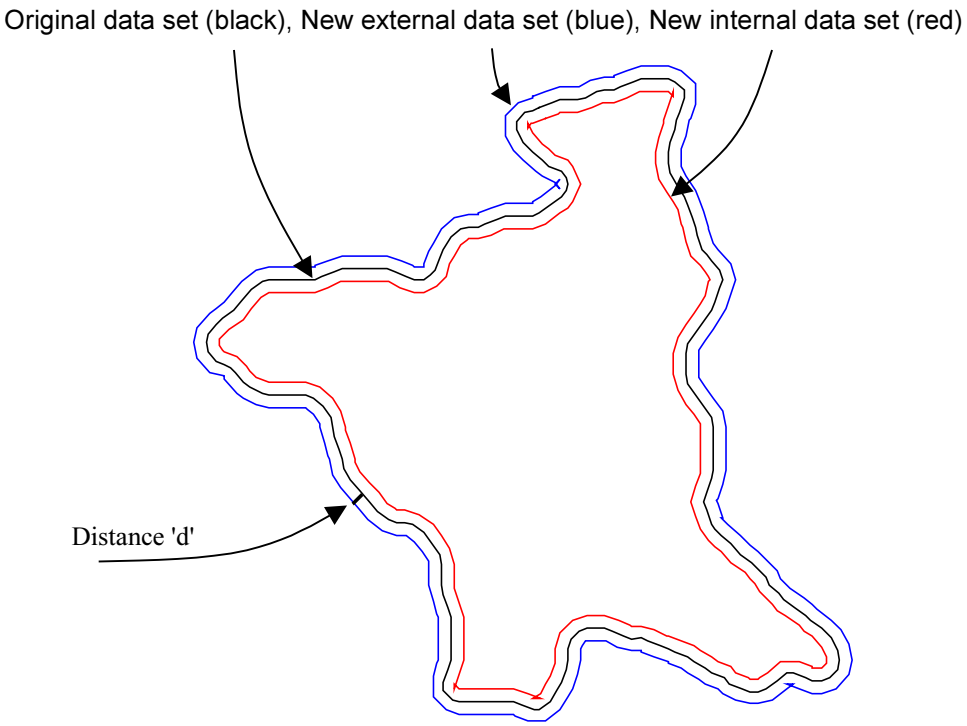


Fig. 2.1: Original data set and new external and internal data sets

In this section, we describe the original data set points and the two added sets (internal and external) as a single set of 3N points. The first N points are the original data points (points 1...N), the second group of N points (N+1,...,2N) are the external points, and the third group of N points (2N+1,...,3N) are the internal points.

With N original data set points and r polynomial coefficients we obtain $3N$ equations and r variables.

The goal is to minimize the total squared-error E:

$$E = \underbrace{\sum_{n=1}^{N}\left(\bar{a}\,\bar{p}^{T}(x_n, y_n)\right)^2}_{\text{Error W.R.T original points}} + \underbrace{\sum_{n=N+1}^{2N}\left(\bar{a}\,\bar{p}^{T}(x_n, y_n)+\varepsilon\right)^2}_{\text{Error W.R.T external points}} + \underbrace{\sum_{n=2N+1}^{3N}\left(\bar{a}\,\bar{p}^{T}(x_n, y_n)-\varepsilon\right)^2}_{\text{Error W.R.T internal points}} \quad (2.7)$$

The constant ε is an arbitrary positive constant.

The error E may be written as:

$$E = \bar{e}\,\bar{e}^{T} \quad (2.8)$$

where,

$$\bar{e} = (\bar{a}M - \bar{b}) \quad (2.9)$$

The $3N^{th}$ dimensional vector $\bar{b}$ and the $r\,x\,3N$ matrix M are defined by:

$$\begin{aligned} \bar{b} &= \left[\bar{0} \quad -\bar{\varepsilon} \quad \bar{\varepsilon}\right] \\ M &= \left[M_0 \quad M_{EX} \quad M_{IN}\right] \end{aligned} \tag{2.10}$$

with,

$$\begin{aligned} M_0 &= \left[\bar{p}^T(x_1, y_1) \quad \ldots \quad \bar{p}^T(x_N, y_N)\right] \\ M_{EX} &= \left[\bar{p}^T(x_{N+1}, y_{N+1}) \quad \ldots \quad \bar{p}^T(x_{2N}, y_{2N})\right] \\ M_{IN} &= \left[\bar{p}^T(x_{2N+1}, y_{2N+1}) \quad \ldots \quad \bar{p}^T(x_{3N}, y_{3N})\right] \end{aligned} \tag{2.11}$$

The vectors $\bar{\varepsilon},\bar{0}$ making up the vector $\bar{b}$ are constant row vectors of length N, with the elements being $\varepsilon,0$, respectively.

Using the *least squares* (LS) solution for this set of equations results in:

$$\bar{a}_{LS} = \bar{b} M^T \left(MM^T\right)^{-1} \tag{2.12}$$

This solution is feasible when the system of equations in (2.12) is over-determined, i.e., there are more equations than parameters.

The parameter vector $\bar{a}_{LS}$ is the best parameters vector (in the LS error sense) of a polynomial, whose zero-set approximates the location of the original data set, and whose values on both sides of the original data set (at distance 'd') are approximately $\pm\varepsilon$ (positive on the outside and negative on the inside).

2.3 Applications of implicit polynomials

2.3.1 Object recognition

Implicit polynomials are currently being used mainly for object recognition. Geometric invariants [11] allow recognition of objects that have undergone affine transformations [12]. Mutual invariants allow the usage of interaction between objects for recognition applications [13,14,15]. Since reliable recognition can only be performed when the mathematical model describing the data (e.g. implicit polynomials) provide an accurate description of the data. Loose fit or high sensitivity to data noise lead to unreliable recognition using invariants calculated from polynomial coefficients.

Other applications in computer graphics, requiring an implicit description of 3D surfaces can also significantly benefit from the existence of a tight and robust fitting algorithm.

In this work we examine the application of implicit polynomials for contour coding, which is useful for image compression.

2.3.2 Contour coding

A basic stage in a shape-adaptive image coding system is contour coding which aims to encode the boundary of the object by following its outline. The leading and most accepted methods for contour coding are chain coding, differential chain coding, B-splines and polygons [16,17].

This work presents a new approach to contour coding, based on an improved fitting of high-degree (up to 18^{th} degree) implicit polynomials which is robust to coefficient quantization.

Implicit polynomials are global models with good representation power of complex object boundaries in 2D images and surfaces in 3D range data [4,7,18,19].

The reasons we have selected Implicit Polynomials for shape representation are as follows:

- Ability to represent a complete object boundary by using a small number of parameters. For example, in a fourth-degree polynomial there are 15 independent coefficients for 2D boundaries and 35 coefficients for 3D surfaces.
- They can represent complex shapes including objects consisting of a few disconnected components, objects that intersect them-selves and objects with holes.
- The proposed algorithm for fitting is robust to noise and to coefficient quantization.

This work is connected with a general scheme for object based coding which consists of the following stages:

1) Finding the object boundaries by using a segmentation algorithm.
2) Fitting geometric models to each object boundary. The models suggested here are of implicit polynomials, depending on the boundary complexity. Complex boundaries are segmented into several curves and are represented using the values of the coefficients of its fitting implicit polynomial model. Using the model with the known values of the coefficients generates a zero-set of the implicit polynomial. The polynomial is constructed in such a way that this zero-set coincides with the data boundary.

3) Compression is obtained by the quantization process of the coefficients converting them from reals to integers (for example, 8 bits for each coefficient). The zero-set fitting generated by the quantized coefficients is the reconstructed data after the compression process.

We begin with a ***segmented image***, according to segments whose regions are easily coded (at a low bit count). We find curves that are shared by each two neighboring segments, code each curve using implicit polynomials and than represent the coefficients of these polynomials with integer values which, along with some side information, are our code words.

In order to achieve good compression results, further segmentation of the curves in the image is often needed. The next step in the coding scheme is therefore curve segmentation, in which several IPs, possibly of different order, are fitted to parts of the curve. The results of this stage are curve segments to which IPs can be fitted and can be efficiently coded.

Finally, implicit polynomials are fitted to the curve segments. This stage uses a fitting algorithm that was developed in order to produce stable polynomials whose coefficients can be quantized using the smallest number of bits, causing the lowest possible distortion. These polynomials are fitted under such constraints that allow data restoration from the quantized coefficients.

As a performance benchmark we use the popular *chain-coding* method for contour coding. This method is described next.

Chain coding

A popular method for contour coding is called *chain coding* [20]. We compare the results of IP coding to chain coding in order to evaluate the effectiveness of the algorithm proposed in this work.

A chain-code codes the relative direction between neighboring boundary pixels. The boundary to be coded has to be continuous, with neighboring pixels at a distance of 1 pixel. There are two main alternatives for using chain codes:

- 4 Neighbors chain coding
- 8 Neighbors chain coding

Using the first option – 4 neighbors chain coding – the direction between each two pixels is coded using two bits, allowing 4 possible directions: up, down, left, right. Therefore, neighboring pixels should be one next to the other and not in diagonals.

Using the second option – 8 neighbors chain coding – the direction between each two pixels is coded using three bits, allowing 8 possible directions: up, down, left, right and four diagonals. Therefore, neighboring pixels should be one next to the other or in diagonal.

Fig. 2.2 shows the possible location of neighboring pixels relative to the current pixel marked as "C".

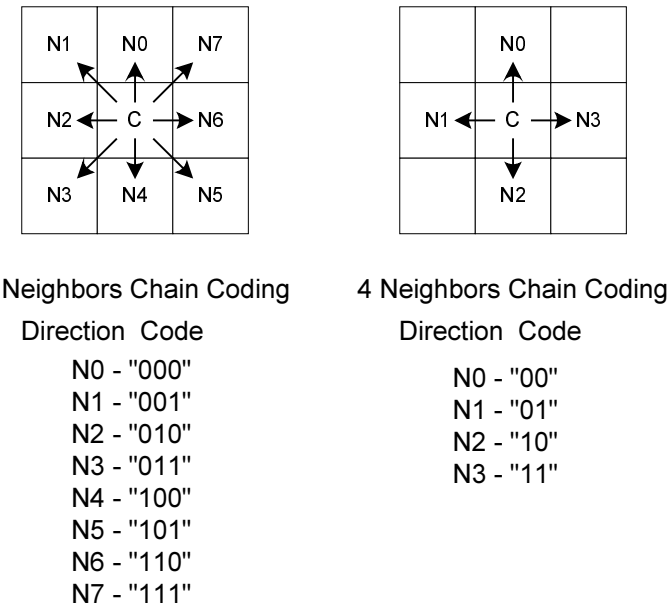


Fig. 2.2: Neighboring pixels coded using 4 and 8 neighbors chain code

There are contours that are more efficiently coded using one of the chain-codes rather than the other. Without the usage of statistical coding, straight horizontal or vertical lines, for example, are most efficiently coded using 4 neighbors chain coding, where the total rate of the code would be $2L$ bits for a line of length L, and $3L$ bits for the same line using 8 neighbors chain coding. A 45° diagonal line is most efficiently coded using 8 neighbors chain coding. For this line, a 8 neighbors chain code requires $\frac{3L}{\sqrt{2}}$ bits and a 4 neighbors chain code requires $\frac{4L}{\sqrt{2}}$ for a line of length L.

Chain codes supply the relative direction between boundary pixels. In order to specify absolute image coordinates for a boundary, a starting point should be

supplied. To properly decode a chain code, the end of the chain must also be specified. This can be accomplished by several means such as specifying the length of a chain, or an ending coordinate.

Since chain codes represent the exact location of the pixels this is a loss-less code. Boundaries restored from chain codes have zero distortion relative to the un-coded boundaries.

Application of entropy coding to the chain code can greatly reduce the required number of code bits. Straight horizontal, vertical or 45° lines have repeating code words and are therefore coded using only a few bits. Lines going at arbitrary angles or free forms are less efficiently coded, but some reduction in the required bit count is usually achieved.

Neighboring objects in an image share contour segments. Apart from objects located at the edges of an image, each boundary point appears in the contour of two objects, therefore creating redundancy in the boundary data of the image. Generalized chain codes [21] take advantage of this fact and use some side information (allowing the association of contour segments to the different objects in the image) to code each boundary point only once. This technique reduces the redundancy of the data, at the expense of additional side information. We also relate to this property of the contour data in the context of implicit polynomial representation in 6.1.

This page intentionally left blank

Chapter 3 Robust polynomial fitting

3.1 Introduction

In this chapter we construct an algorithm for fitting implicit polynomials to contour data. We begin this chapter with the formulation of error bounds for the error in the location of polynomial zero-set points due to coefficients quantization. We than apply the error bounds that are formulated here to the fitting algorithms presented in section 2.2. As a result, we find that the *3L* algorithm can be improved by minimizing not only the distance between the zero-set and the data points but also the sensitivity of the resulting zero-set to changes in the coefficients. We denote two versions of this algorithm as – *Min-Max* and *Min-Var*, corresponding to the optimized property.

The layout of this chapter is as follows: section 3.2 includes a study of the sensitivity of the zero-set relative to changes in the coefficients of the polynomial, and the resulting error properties. The sensitivity of the algorithms presented in section 2.2 is calculated in section 3.3. In section 3.4 we construct our fitting algorithm based on the results of section 3.2. Simulation results are provided in section 3.5, and the chapter is summarized in section 3.6.

3.2 Zero-set sensitivity to parameter changes

When the values of the coefficients in the parameter vector change, the entire zero-set changes. In this section we examine how changes in the parameter vector affect the location of a point on the zero-set. Since the zero-set is continuous, we cannot measure the distance between two points (before and after a parameter change). We define the change in a zero-set point ($d\bar{z}$) as the distance between an original zero-set point, $\bar{z} = (z_x, z_y)$, and the closest point on the new data set (following the change in the parameter vector).

In this section, we formulate the error and sensitivity functions at point $\bar{z}(x, y) = (z_x, z_y)$.

The sensitivity function $\overline{S}_{\bar{a}}^{\bar{z}}(x, y)$ is a $2 \times r$ matrix defined by:

$$S_{\bar{a}}^{\bar{z}}(x,y)=\frac{d\bar{z}(x,y)}{d\bar{a}} \tag{3.1}$$

This function describes the relation between small changes (errors) in the coefficients and small changes in the location of zero set points.

The change in the location of a zero-set point $\bar{\varepsilon}_Z(x,y)=[\bar{\varepsilon}_X \quad \bar{\varepsilon}_Y]$ resulting from a small change in the parameters $\bar{\varepsilon}_a=[\varepsilon_{a_1} \quad ... \quad \varepsilon_{a_r}]$ is the product of the error components with the associated sensitivity function[1]:

$$\bar{\varepsilon}_Z(x,y)\cong \bar{S}_{\bar{a}}^{\bar{z}}(x,y)\bar{\varepsilon}_a{}^T \tag{3.2}$$

3.2.1 Sensitivity function

Small changes in the position of the zero-set, in the tangent direction, move zero-set points back into the zero-set, therefore for the purpose of evaluating zero-set changes, it is sufficient to examine changes in the direction perpendicular to the zero-set. We denote $u(x,y)$ as the component of $\bar{z}(x,y)$ locally perpendicular to the zero-set (see Fig. 3.1):

$$u(x,y)=\langle \bar{z}(x,y)\cdot \nabla P_{\bar{a}}(x,y)\rangle \tag{3.3}$$

Therefore, the sensitivity function of interest to us is given by the vector $S_{\bar{a}}^{u}(x,y)=\frac{du(x,y)}{d\bar{a}}$, and denotes the relation between changes in the zero-set, locally perpendicular to the zero-set, and changes in the coefficient vector.

This function can be written as a product of two independent parts:

$$\bar{S}_{\bar{a}}^{u}(x,y)=\frac{du}{dP_{\bar{a}}(x,y)}\frac{dP_{\bar{a}}(x,y)}{d\bar{a}} \tag{3.4}$$

where $P_{\bar{a}}(x,y)$ is the value of the polynomial described by $\bar{a}$ at location (x,y).

The right-hand part of the product in (3.4) describes the change in the value of the polynomial, at the location of the original zero-set point, due to a small change in the parameter vector. This part is a vector (has an element for each of the elements in $\bar{a}$). The left-hand part describes the deviation of the zero-set point in the direction perpendicular to the zero-set due to a small change in the value of the polynomial, at the location of the original zero-set point. This part is a scalar.

---

[1] This is a 1st order tailor approximation. The accuracy of the approximation depends on the magnitude of the error components.

We shall evaluate each part of the sensitivity function in (3.4) separately, beginning with the left-hand part.

The deviation in the location of a zero point in a direction locally perpendicular to the zero-set, following a change in the parameter vector, is described in Fig. 3.1.

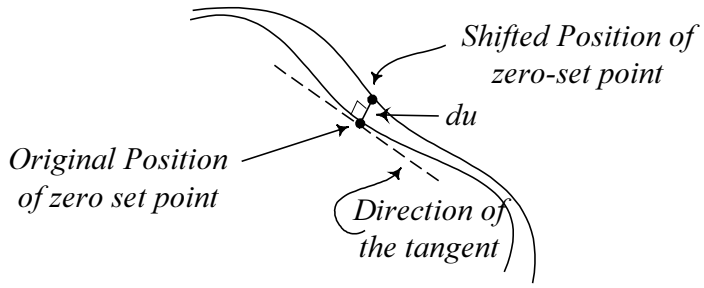


Fig. 3.1: Location of a zero-set point before and after a small change in the coefficients

For small changes in the parameter vector the ratio between the position error in the perpendicular direction du and the change in the value of the function at point (x, y) is the inverse of the gradient of $P_{\bar{a}}(x, y)$:

$$\frac{du}{dP_{\bar{a}}}(x, y) = \frac{1}{\left\|\nabla P_{\bar{a}}(x, y)\right\|} = \frac{1}{\sqrt{\left(\frac{\partial P_{\bar{a}}(x, y)}{\partial x}\right)^2 + \left(\frac{\partial P_{\bar{a}}(x, y)}{\partial y}\right)^2}} \tag{3.5}$$

The right-hand part of the sensitivity function $\frac{dP_{\bar{a}}(x, y)}{d\bar{a}}$ can be directly calculated from $P_{\bar{a}}(x, y) = \bar{a}\,\bar{p}^T(x, y)$ (where $\bar{p}(x, y)$ is the monomial vector), and results in:

$$\frac{dP_{\bar{a}}(x, y)}{d\bar{a}} = \bar{p}(x, y) \tag{3.6}$$

Using (3.5) and (3.6) the sensitivity function in (3.4) can now be written as:

$$\bar{S}^u_{\bar{a}}(x, y) = \frac{\bar{p}(x, y)}{\left\|\nabla P_{\bar{a}}(x, y)\right\|} \tag{3.7}$$

3.2.2 Properties of the zero-set errors

Now, we can examine the resulting fitting errors due to small changes in the coefficients and derive some useful properties of these errors.

The error in the direction perpendicular to the zero-set $\varepsilon_u(x, y)$ is:

$$\varepsilon_u(x, y) = \bar{S}^u_{\bar{a}}(x, y) \cdot \bar{\varepsilon}_a{}^T = \frac{\bar{p}(x, y)}{\left\|\nabla P_{\bar{a}}(x, y)\right\|} \bar{\varepsilon}_a{}^T = \frac{\sum_{k=1}^{r} p_k(x, y)\varepsilon_{a_k}}{\left\|\nabla P_{\bar{a}}(x, y)\right\|} \tag{3.8}$$

For a given point (x,y) on the zero set, the maximal error is bound by:

$$\left|\varepsilon_u(x,y)\right| \le \frac{\left|\sum_{k=1}^{r} p_k(x,y)\varepsilon_{a_k}\right|}{\left\|\nabla P_{\bar{a}}(x,y)\right\|} \le \frac{\sum_{k=1}^{r}\left|p_k(x,y)\right|\left|\varepsilon_{a_k}\right|}{\left\|\nabla P_{\bar{a}}(x,y)\right\|} \le \varepsilon_{MAX}\frac{\sum_{k=1}^{r}\left|p_k(x,y)\right|}{\left\|\nabla P_{\bar{a}}(x,y)\right\|} \tag{3.9}$$

where $\varepsilon_{MAX} = \max\{abs(\varepsilon_{a_k})\}$.

When components of the parameter vector are independent random variables with zero mean, the variance of $\varepsilon_u(x,y)$ is calculated by:

$$\mathrm{var}(\varepsilon_u(x,y)) = \frac{\mathrm{var}\left(\sum_{k=1}^{r} p_k(x,y)\varepsilon_{a_k}\right)}{\left\|\nabla P_{\bar{a}}(x,y)\right\|^2} = \frac{\sum_{k=1}^{r}\left((p_k(x,y))^2\,\mathrm{var}(\varepsilon_{a_k})\right)}{\left\|\nabla P_{\bar{a}}(x,y)\right\|^2} \tag{3.10}$$

and when all the error components have the same variance ($\mathrm{var}(\varepsilon_K) = \sigma_\varepsilon^2$), we obtain:

$$\mathrm{var}(\varepsilon_u(x,y)) = \sigma_\varepsilon^2 \frac{\sum_{k=1}^{r} {p_k}^2(x,y)}{\left\|\nabla P_{\bar{a}}(x,y)\right\|^2} \tag{3.11}$$

3.3 Analysis of existing algorithms

In this section we use the results of section 3.2 to analyze the error properties of solutions achieved by Taubin's and the *3L* algorithms described in section 2.2.

3.3.1 Analysis of the non-linear algorithm

The non-linear algorithm (see section 2.2.1) has a cost function that aims at reducing the distance between the data set points and the zero set, but does not consider the sensitivity of the obtained zero-set to coefficient errors.

The sensitivity of the zero-set to small changes in the coefficients is governed by the values of the gradient of the polynomial at the location of the data points. Since the algorithm does not attempt to set values to the gradient, we need to find a different technique for calculating the error bounds for the zero-set of the resulting polynomials.

We can write the quantizaed coefficients $\bar{a}_Q$ vector as the sum of the optimal coefficients vector and the error components in the direction of each eigenvector as:

$$\bar{a}_Q = \bar{a}_{OPT} + \sum_{i=1}^{i=r}\langle \bar{\varepsilon}_a \bar{v}_i \rangle \bar{v}_i \tag{3.12}$$

Since quantization errors are independent of the data generating the matrix SM, the generated error vector has components in the directions of all the eigenvectors of SM.

The error is therefore:

$$E = \bar{a}_Q \, SM \, \bar{a}_Q{}^T = \lambda_{MIN}{}^2 + \sum_{i=1}^{r} \left(\langle \bar{\varepsilon}_a \bar{v}_i \rangle \lambda_i \right)^2 \tag{3.13}$$

and can be bound by:

$$\begin{aligned} E &\le \lambda_{MIN}{}^2 + \left\| \bar{\varepsilon}_a \right\|^2 \lambda_{MAX}{}^2 = \lambda_{MIN}{}^2 \left(1 + \left(\left\| \bar{\varepsilon}_a \right\| CN(SM) \right)^2 \right) \\ &\le \lambda_{MIN}{}^2 \left(1 + \left(2^{-N_{BITS}} \, CN(SM) \right)^2 \right) \end{aligned} \tag{3.14}$$

where λ_{MAX} is the maximal eigenvalue, and $CN(SM)$ is the condition number of SM.

In order to obtain a good fit we need to allocate:

$$N_{BITS} \ge \log_2 CN(SM) \tag{3.15}$$

In our simulations the condition number of SM was between 2^{20} and 2^{26}.

3.3.2 Analysis of the 3L algorithm

In section 3.2 we analyzed the sensitivity of the zero-set to changes in the coefficient vector. This analysis holds for points on the zero-set of the polynomial (before coefficient change).

When the fitting of an IP to data is good, the value of the polynomial at the data points is close to zero, and instead of checking the sensitivity at the position of the zero-set, the sensitivity may be examined at the data points. This substitution allows the analysis of the maximal error resulting from coefficient changes without having to find the zero-set of the polynomial, which represents coded data points. Of course, this substitution can be make only when the fitting is tight[2].

Therefor, assuming the *3L* algorithm produces tight fitting results, and according to (3.9), the error for each data set points n is limited by:

[2] Analysis of quantization errors is required only when the original unquantized fitting is tight. Loose fitting is unacceptable for coding and therefore we are not interested in it's sensitivity to quantization.

$$\left|\varepsilon_u\left(x_n, y_n\right)\right| \leq \varepsilon_{MAX} \frac{\sum_{k=1}^{r}\left|p_k\left(x_n, y_n\right)\right|}{\left\|\nabla P_{\bar{a}}\left(x_n, y_n\right)\right\|} \tag{3.16}$$

As described in section 2.2.2, the expansion and shrinking operation of the *3L* algorithm (when done very tightly around the original data set) is equivalent to differentiation of the polynomial. According to the *3L* algorithm, a constant values of the polynomial ($\pm\varepsilon$) is required at a constant distance from the data set (d). This implies a demand for constant derivatives in the direction perpendicular to the data set, leading to a constant gradient near the data set points - $\left\|\nabla P_{\bar{a}}\left(x_n, y_n\right)\right\| = K_1 \quad \forall n = 1,...,N$, where K_1 is a positive constant.

The maximal error for each data points is therefore limited by:

$$\left|\varepsilon_u\left(x_n, y_n\right)\right| \leq \underbrace{\frac{\varepsilon_{MAX}}{\left\|\nabla P_{\bar{a}}\left(x_n, y_n\right)\right\|}}_{\text{Constant value} = K_2} \sum_{k=1}^{r}\left|p_k\left(x_n, y_n\right)\right| = K_2 \sum_{k=1}^{r}\left|p_k\left(x_n, y_n\right)\right| \tag{3.17}$$

where

$$K_2 = \frac{\varepsilon_{MAX}}{\left\|\nabla P_{\bar{a}}\left(x_n, y_n\right)\right\|} \tag{3.18}$$

It is clear that the *3L* algorithm yields different error bounds for different data points, depending on the different values of the monomial vector $\bar{p}\left(x_n, y_n\right)$. In section 3.4 we derive an algorithm which yields constant error bounds for all data points.

3.4 Stable Fitting Algorithm

In this section we use the results of section 3.2, with regard to the error properties of the polynomial, to construct a fitting algorithm. Our goal is a polynomial with a zero-set which is as stable as possible, relative to changes in the coefficients.

We begin by modifying the original *3L* algorithm. This modification is the replacement of the expansion and shrinking operations with differentiation of the polynomial. The motivation and the description of this modification are presented in section 3.4.1.

In section 3.4.2 we continue by modifying the cost function for the polynomial in order to achieve a polynomial with some required stability properties.

Section 3.4.3 describes the necessary expansions for 3D polynomial fitting.

3.4.1 Modification of the 3L algorithm

In order to achieve a solution that tightly fits the data, we need to generate the internal and external data sets close to the original data points. This implies that for tight fitting results ε and d (defined and used in section 2.2.2) should approach zero.

If we attempt to satisfy this demand and examine the results of the *least squares* solution $\left(\overline{a}_{LS} = \overline{b}M^T\left(MM^T\right)^{-1}\right)$, we would notice that the factor $M^T\left(MM^T\right)^{-1}$ heads to ∞ while $\overline{b}$ goes to zero. Small numerical errors when inverting the matrix $\left(MM^T\right)$ or when creating the internal and external data sets are manifested by large fitting errors.

Fitting attempts have shown that it is possible to choose the parameter d as about 5% of the geometric size of the object described by the data and obtain stable results. Smaller values of d (which are desirable for obtaining tighter fitting) yield less stable fitting results.

Choosing large values for the parameter d can also cause problems as seen in Fig. 3.2:

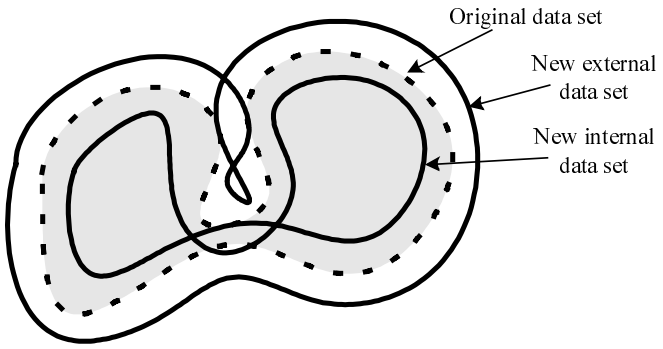


Fig. 3.2: Expansion and shrinking result with a large d relative to the object size

As we can see from this example, choosing a large value of d may lead to the generation of conflicting requirements, caused by overlapping sets (original, internal and external), therefore leading to poor fitting results.

We describe here a method for replacing the added sets by explicit differentiation of the polynomial. Motivation for this replacement is given below.

The basic solution, including the expanding and shrinking of the original data points and the solution of (2.12) is actually an implementation of an approximated differentiation. Fig. 3.3 demonstrates the location of the added internal and external sets and the values of the polynomial at these sets. This figure also shows how the

required values generate an implicit requirement for the value of the derivative of the polynomial in the direction perpendicular to the original data set.

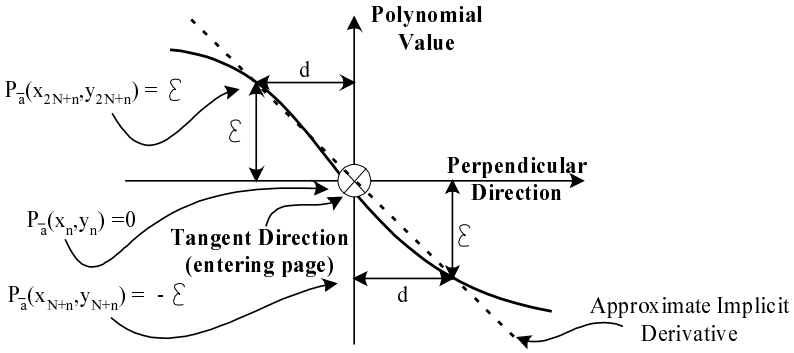


Fig. 3.3: View of added internal and external points from the direction of the tangent at (x_n, y_n)

By replacing the numerical approximation of the differentiation operation (implemented via the expansion and shrinking of the data) with an explicit differentiation of the polynomial we avoid the numerical difficulties in the original *3L* algorithm.

We begin by calculating the differentials of the monomial in both axes. The vectors $\bar{p}_X(x_n, y_n), \bar{p}_Y(x_n, y_n)$ describe the derivatives with respect to x and y, respectively, of the monomial vector at point (x_n, y_n):

$$\begin{aligned} \bar{p}_X(x_n, y_n) &= \frac{d}{dx}\bar{p}(x,y)\Big|_{(x=x_n, y=y_n)} \\ \bar{p}_Y(x_n, y_n) &= \frac{d}{dy}\bar{p}(x,y)\Big|_{(x=x_n, y=y_n)} \end{aligned} \tag{3.19}$$

Multiplying the derivatives of the monomials by the coefficient vector yields the respective derivatives of the polynomial according to:

$$\begin{aligned} \frac{d}{dx}P_{\bar{a}}(x,y)\Big|_{(x=x_n, y=y_n)} &= \bar{a}\,\bar{p}_X{}^T(x_n, y_n) \\ \frac{d}{dy}P_{\bar{a}}(x,y)\Big|_{(x=x_n, y=y_n)} &= \bar{a}\,\bar{p}_Y{}^T(x_n, y_n) \end{aligned} \tag{3.20}$$

We attempt to bring the polynomial differential at the location of the original data set points to the direction of the line locally perpendicular to the data set near each data set point (see Fig. 3.4).

The angle relative to the X axis of the perpendicular vector is denoted by α_n, as shown in Fig. 3.4, which also shows that in order to calculate α_n, we fit a straight line (1st degree polynomial) to the points around data point (x_n, y_n).

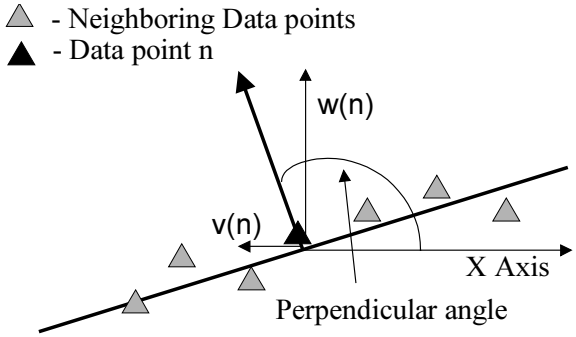


Fig. 3.4: Fitting result of a 1st degree polynomial to 7 points about point (x_n, y_n)

The vectors $\overline{v} = [v_1, ..., v_N]$, $\overline{w} = [w_1, ..., w_N]$ (where N is the number of points in the data set) contain the data of the perpendicular vectors for each of the data set points. Each pair of elements (v_n, w_n) in $\overline{v}, \overline{w}$ make up a unit vector pointing at the direction locally perpendicular to the data set at each point n.

Having estimated α_n from the data set, at each point n, we can calculate the vectors $\overline{v}, \overline{w}$ from the relations:

$$\left.\begin{array}{l} \dfrac{v_n}{w_n} = tg(\alpha_n) \\ \sqrt{\left(v_n{}^2 + w_n{}^2\right)} = \dfrac{\varepsilon}{d} \end{array}\right\} \Rightarrow \begin{array}{l} v_n = \dfrac{\varepsilon}{d}\sin(\alpha_n) \\ w_n = \dfrac{\varepsilon}{d}\cos(\alpha_n) \end{array} \tag{3.21}$$

In order to have the gradient of the polynomial point in the direction of the perpendicular to the data set, with a value of $\dfrac{\varepsilon}{d}$ (as seen in Fig. 3.3), and keep the value of the polynomial at the location of the data points equal to zero, the expressions for $\overline{b}$ and M are now:

$$\begin{array}{l} \overline{b} = \left[\overline{0} \quad \overline{v} \quad \overline{w}\right] \\ M = \left[M_0 \quad M \quad M\right] \end{array} \tag{3.22}$$

where,

$$
\begin{aligned}
M_0 &= \left[\bar{p}\ \ (x_1, y_1)\ \ \ldots\ \ \bar{p}\ \ (x\ \ , y\ \ )\right] \\
M\ \ &= \left[\bar{p}\ \ (x_1, y_1)\ \ \ldots\ \ \bar{p}\ \ (x\ \ , y\ \ )\right] \\
M\ \ &= \left[\bar{p}\ \ (x_1, y_1)\ \ \ldots\ \ \bar{p}\ \ (x\ \ , y\ \ )\right]
\end{aligned}
\tag{3.23}
$$

Using these relations to solve (2.9), the form of the *least squares* solution remains $\bar{a}_{LS} = \bar{b} M^T \left(MM^T\right)^{-1}$.

Since the required value of the gradients at each data point is a constant $\left(\frac{\varepsilon}{d}\right)$, this value can be extracted from the expression according to:

$$
\bar{a}_{LS} = \frac{\varepsilon}{d} \bar{b}_{norm} M^T \left(MM^T\right)^{-1} \tag{3.24}
$$

where the vector $\bar{b}_{norm}$ contains the elements of $\bar{b}$ normalized to a magnitude of 1:

$$
\left.\begin{array}{l} \dfrac{v_n}{w_n} = tg(\alpha_n) \\ \sqrt{\left(v_n{}^2 + w_n{}^2\right)} = 1 \end{array}\right\} \Rightarrow \begin{array}{l} v_n = \sin(\alpha_n) \\ w_n = \cos(\alpha_n) \end{array} \tag{3.25}
$$

This scaling does not affect the location of the zero-set of the polynomial and is therefore permissible. When allocating bits for each coefficient, we scale the coefficients so that the largest coefficient equals 1. Since scaling is involved, we ignore the constant $\left(\frac{\varepsilon}{d}\right)$, and simplify notation. This action also demonstrates that both constants used for the *3L* algorithm (ε and d) are insignificant when implementing implicit differentiation instead of the expansion and shrinking operations.

3.4.2 Using the zero-set error properties to derive a fitting algorithm

In section 3.2 we analyzed the changes in the location of zero-set points resulting from changes in the parameters. We now show that a fitting algorithm which produces low zero-set sensitivity to coefficient changes, is also useful for obtaining a tight fit (without coefficient errors).

The problem of polynomial fitting is over-determined. I.e., there are more requirements than variables (coefficients). Therefore, for all but very singular problems, all the requirements cannot be met. Instead, the requirements are approximated so that a cost function is minimized.

Small groups of data points can be selected (out of all the data points) so that the fitting problem can be exactly solved. For each of these group of points there exists a locally optimal coefficient vector, which produces a polynomial whose zero-set passes though the data points and satisfies any additional requirements (typically, values of derivatives at data points). We construct such group of points around each point n, and obtain the polynomial, which exactly solves the set of equations associated with this group of points. We denote the coefficient vector of this polynomial as $\bar{a}_{n-LS}$.

We consider the coefficient vector, of the polynomial describing all the data points (obtained by any fitting algorithm) as the sum of the locally optimal parameter vector and added error.

$$\bar{a}_{n-LS} = \bar{a}_n + \left(\bar{a}_{LS} - \bar{a}_n\right) \tag{3.26}$$

It is clear that reducing the sensitivity to coefficient errors would reduce the zero-set error caused by this type of error.

Therefore we can use the above analysis by substituting zero-set points with the given data set points in order to calculate the expected properties of the fitting result.

In order to obtain the best parameter vector we need to define a criterion according to which we would perform the optimization process.

We are interested in two properties – a) best fit to the data, b) minimal deviation due to changes in the coefficients. These demands can be stated as follows:

Find $\bar{a}$ so that the following requirements hold for each of the data set points $\left(x_n, y_n\right)$:

a) The value of the polynomial is zero on each of the data set points: $P_{\bar{a}}\left(x_n, y_n\right) = 0$.

b) Errors generated due to changes in the coefficients are minimal:

Using the error bound derived in section 3.2.2 and limiting the coefficient changes to ε_{MAX}, we obtain a bound on the maximal zero-set error for each point. We look for a polynomial which leads to minimal zero-set errors, i.e. $\dfrac{\varepsilon_{MAX}\sum_{k=1}^{r}\left|p_k\left(x_n, y_n\right)\right|}{\left\|\nabla P_{\bar{a}}\left(x, y\right)\right\|}$ should be minimal for each of the data set points.

The first requirement also implies that the tangent direction of the polynomial equals the tangent direction of the data for each of the data set points. This is due to

the fact that the gradient of the polynomial (or any other implicit description) is perpendicular to the zero-set. This leads to the next requirement:

$$\frac{\partial P_{\bar{a}}(x_n, y_n)}{\partial y} \Big/ \frac{\partial P_{\bar{a}}(x_n, y_n)}{\partial x} = tg(\alpha_n) \tag{3.27}$$

where α_n is the angle of the local perpendicular to the data set about point n whose location is (x_n, y_n) - see Fig. 3.4.

We denote the coefficient vector that best achieves these requirements as $\bar{a}_{OPT}$.

Since no data point has priority over another (if no weighting is used), we can limit the maximal fitting error due to changes in the coefficients to a constant value by requiring that the expression $\dfrac{\varepsilon_{MAX} \sum_{k=1}^{r} |p_k(x_n, y_n)|}{\|\nabla P_{\bar{a}}(x, y)\|}$ would be equal for values of $n = 1,..., N$. Since the constant is insignificant for the optimization, the following is required:

$$\|\nabla P_{\bar{a}}(x, y)\| = \sum_{k=1}^{r} |p_k(x_n, y_n)| \quad for\ n = 1,..., N \tag{3.28}$$

The *LS* solution for the requirements presented above can be calculated within the framework of the solution described in section 2.2.2. Modifying (3.25) according to the following yields the solution referred here as ***Min-Max***:

$$\left.\begin{aligned} &\frac{v_n}{w_n} = tg(\alpha_n) \\ &\sqrt{(v_n^{\,2} + w_n^{\,2})} = \sum_{k=1}^{r} |p_k(x_n, y_n)| \end{aligned}\right\} \Rightarrow \begin{aligned} v_n &= \left(\sum_{k=1}^{r} |p_k(x_n, y_n)|\right) \sin(\alpha_n) \\ w_n &= \left(\sum_{k=1}^{r} |p_k(x_n, y_n)|\right) \cos(\alpha_n) \end{aligned} \tag{3.29}$$

Using the same formulation, we can implement a criterion that would minimize the variance of the error (over the elements of the error components). Modifying (3.25) as follows yields the solution referred as ***Min-Var***:

$$\left.\begin{aligned} &\frac{v_n}{w_n} = tg(\alpha_n) \\ &\sqrt{(v_n^{\,2} + w_n^{\,2})} = \sqrt{\sum_{k=1}^{r} p_k^{\,2}(x_n, y_n)} \end{aligned}\right\} \Rightarrow \begin{aligned} v_n &= \sqrt{\sum_{k=1}^{r} p_k^{\,2}(x_n, y_n)} \sin(\alpha_n) \\ w_n &= \sqrt{\sum_{k=1}^{r} p_k^{\,2}(x_n, y_n)} \cos(\alpha_n) \end{aligned} \tag{3.30}$$

Adding weights to data points

The above formulation was derived with the assumption that all data points have the same priority. Prioritizing the data points (adding weights to the errors) yields the following cost function:

$$E_W = \bar{e}_W \bar{e}_W^T \tag{3.31}$$

$$\bar{e}_W = \left(\bar{a}M - \bar{b}\right)W = \bar{a}MW - \bar{b}W \tag{3.32}$$

where W is a *diagonal* weighting matrix whose diagonal elements are the relative weights of the corresponding data points. The diagonal elements $1,...,n$ are the weights for the zero fitting, and elements $n+1,...,2n$ and $2n+1,...,3n$ are the weights for the sensitivity.

Substituting $\bar{a}M \to \bar{a}MW$ and $\bar{b} \to \bar{b}W$ the *LS* solution is:

$$\underline{a}_{WLS} = \bar{b}WW^T M^T \left(MWW^T M^T\right)^{-1} = \bar{b}W^2 M^T \left(MW^2 M^T\right)^{-1} \tag{3.33}$$

Choosing large values for the first n elements prioritizes a good fit with less regard for sensitivity, while choosing large values for the second the third n elements prioritizes a stable fit on the expense of initial accuracy.

By using different weights for different data points, we can achieve tighter fitting at points where important image features are present.

3.4.3 Extension to 3D

The algorithm presented in this chapter can be extended into 3D for fitting surfaces of 3D bodies.

The development of the fitting algorithm for 3D bodies follows the exact procedure presented here with the following modifications:

- All coordinates are given in 3D.
- Perpendicular vectors are now calculated as normals to tangent surfaces (instead of normals to lines).

The *LS* solution for the 3D fitting problem is:

$$\bar{a}_{LS} = \bar{b}M^T \left(MM^T\right)^{-1} \tag{3.34}$$

where,

$$\begin{aligned} \bar{b} &= \begin{bmatrix} \bar{0} & \bar{v} & \bar{w} & \bar{q} \end{bmatrix} \\ M &= \begin{bmatrix} M_0 & M_X & M_Y & M_Z \end{bmatrix} \end{aligned} \tag{3.35}$$

and,

$$
\begin{aligned}
M_0 &= \left[\bar{p}^T(x_1,y_1,z_1) \quad \ldots \quad \bar{p}^T(x_N,y_N,z_N)\right] \\
M_X &= \left[\bar{p}_X{}^T(x_1,y_1,z_1) \quad \ldots \quad \bar{p}_X{}^T(x_N,y_N,z_N)\right] \\
M_Y &= \left[\bar{p}_Y{}^T(x_1,y_1,z_1) \quad \ldots \quad \bar{p}_Y{}^T(x_N,y_N,z_N)\right] \\
M_Z &= \left[\bar{p}_Z{}^T(x_1,y_1,z_1) \quad \ldots \quad \bar{p}_Z{}^T(x_N,y_N,z_N)\right]
\end{aligned} \tag{3.36}
$$

$$
\begin{bmatrix} v_n & w_n & q_n \end{bmatrix}^T = g_n \cdot \overline{perp}_n \tag{3.37}
$$

$\overline{perp}_n$ is a unit vector locally perpendicular to the surface near data point n. This vector can be calculated by fitting a 1st order polynomial (plane) to the points around point n. The coefficient vector of that polynomial is in fact the desired vector $\overline{perp}_n$.

g_n is calculated according to the desired algorithm:

$$
\begin{aligned}
g_{n-3L} &= 1 \\
g_{n-MINMAX} &= \sum_{k=1}^{r} \left| p_k(x_n,y_n) \right| \\
g_{n-MINVAR} &= \sqrt{\sum_{k=1}^{r} p_k{}^2(x_n,y_n)}
\end{aligned} \tag{3.38}
$$

The monomial differentials $\bar{p}_X(x_n,y_n,z_n), \bar{p}_Y(x_n,y_n,z_n)$ are as defined for the 2D case and $\bar{p}\,(x_n,y_n,z_n)$ is the differential of the monomial vector in the Z direction for data point n.

3.5 Simulation results

In this section we present simulation results of the fitting algorithms presented in section 2.2.2 (*3L* algorithm) and of the proposed algorithm derived in section 3.4.

In this Section we refer to these algorithms as:

- *3L – 3L* Fitting algorithm in section 2.2.2.
- *Min-Max* – Fitting algorithm derived by using (3.29) in section 3.4 for minimizing the maximal error.
- *Min-Var* – Fitting algorithm derived by using (3.30) in section 3.4 for minimizing the error variance.

The data used here is taken from segmented medical images.

3.5.1 Sensitivity to coefficient quantization

In this section we examine by simulation the sensitivity of the algorithms presented here to coefficient changes. We use uniformly distributed random noise to simulate

the effect of quantization. The results of this test indicate which algorithm yields the most robust fitting results when coefficient quantization is made. Quantization of the coefficients to the closest integer yields only one value of the coefficient error vector and does not allow a statistical analysis of the error properties of the quantization errors.

In order to evaluate the quality of the solution, a statistical analysis has been performed.

Two objects have been tested with the three of the algorithms presented above (*3L, Min-Var, Min-Max*).

Tables 1(a), 1(b) compare the errors obtained using the examined fitting algorithms for the boundaries of two different objects, each quantized with a different number of bits per coefficient.

The same random noise with a uniform distribution of 1 LSB was added to the coefficient vectors obtained via each fitting algorithm. For each algorithm two tests were made:

$$E_{RMS} = \sqrt{\sum_{n=1}^{N} e_n^2}\ ,\ E_{MAX} = \max\{e_1,...,e_n\} \tag{3.39}$$

The mean value and variance (over different noise vectors) of these tests are shown in Table I(a) and Table I(b).

Table I: Comparison of error statistics using *3L*, *Min-Var* and *Min-Max* fitting

(a)

Algorithm →	*3L*		*Min-Var*		*Min-Max*	
Test ↓	Mean	Variance	Mean	Variance	Mean	Variance
E_{RMS}	0.04	0.034	0.02	***0.003***	***0.019***	0.0033
E_{MAX}	0.15	0.22	0.046	***0.007***	***0.044***	0.015

(b)

Algorithm →	*3L*		*Min-Var*		*Min-Max*	
Test ↓	Mean	Variance	Mean	Variance	Mean	Variance
E_{RMS}	0.042	0.001	0.035	***0.0007***	***0.032***	0.0008
E_{MAX}	0.11	0.007	0.097	***0.0063***	***0.079***	0.0088

(c)

Algorithm →	*3L*		*Min-Var*		*Min-Max*	
Test ↓	Mean	Variance	Mean	Variance	Mean	Variance
E_{RMS}	0.0415	0.0021	0.0352	***0.0017***	***0.0317***	0.0017
E_{MAX}	0.1132	0.0258	0.0979	***0.0232***	***0.0795***	0.0348

RMS and maximal errors over boundary points of contour using different polynomial orders and different number of bits per coefficient are shown.
(a): Order – 12, Bits – 16 ; (b): Order – 4, Bits – 6 ; (c): Order – 4, Bits - 8

By examining the three tables above, it is evident that the use of different criterion functions is effective for obtaining different goals.

The original *3L* algorithm does not define any stability goal and therefore achieves no best score in any test.

For both objects, the mean error is minimal when using the *Min-Max* algorithm. This result is expected because the criterion demands the lowest maximal error and therefore also leads to a minimal mean error.

In parallel, for both bodies, the error variance is lowest by using the *Min-Var* algorithm. This result is also predictable because the criterion used minimizes this quantity (when averaging over different noise vectors as done in the simulation).

3.5.2 Plots of sensitivity function

The intention of the simulations presented in this section is to graphically demonstrate the effects of the values of the sensitivity function obtained by the implementation of the different algorithms presented in this chapter.

The boundary in Fig. 3.5 was used for the simulations in this section. We used a relatively simple shape, which allows a clear illustration of the properties of fitting errors resulting from coefficient quantization.

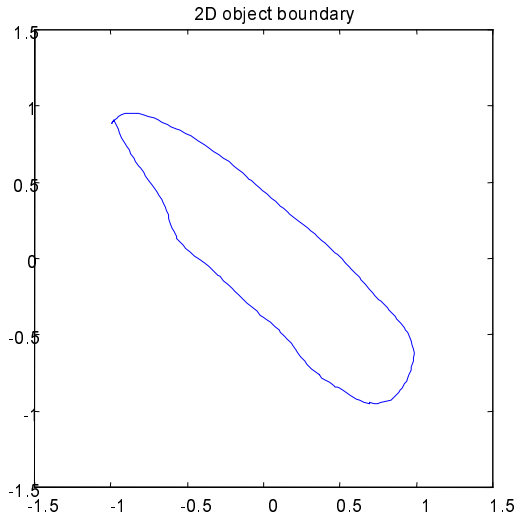


Fig. 3.5: Object boundary for sensitivity analysis

The values of the sensitivity function for each point of the data set was calculated, using the polynomials obtained by the different fitting algorithms. The fitting results of 8^{th} order polynomials and the values of the sensitivity function are plotted in Fig. 3.6(a1, b1, c1). The sensitivity function determines the change in the location of the zero-set as a result of changes in the coefficient, in the direction perpendicular to the zero-set. Therefore, the value of the sensitivity function was plotted as vectors perpendicular to the zero-set, with a length proportional to the value of the function. The value of the sensitivity value was normalized (divided by 210,000) for all three rows to obtain a clear image of the relative sensitivity obtained using the different fitting algorithms. As seen in Fig. 3.6, all three algorithms produced good fitting results using un-quantized coefficients. This is due to the fact that the object's boundary is relatively simple for an 8^{th} order polynomial. However, the *3L* and

Min-Var algorithms yield sensitivity function values that are much greater in some parts of the object than the values produced by the *Min-Max* algorithm.

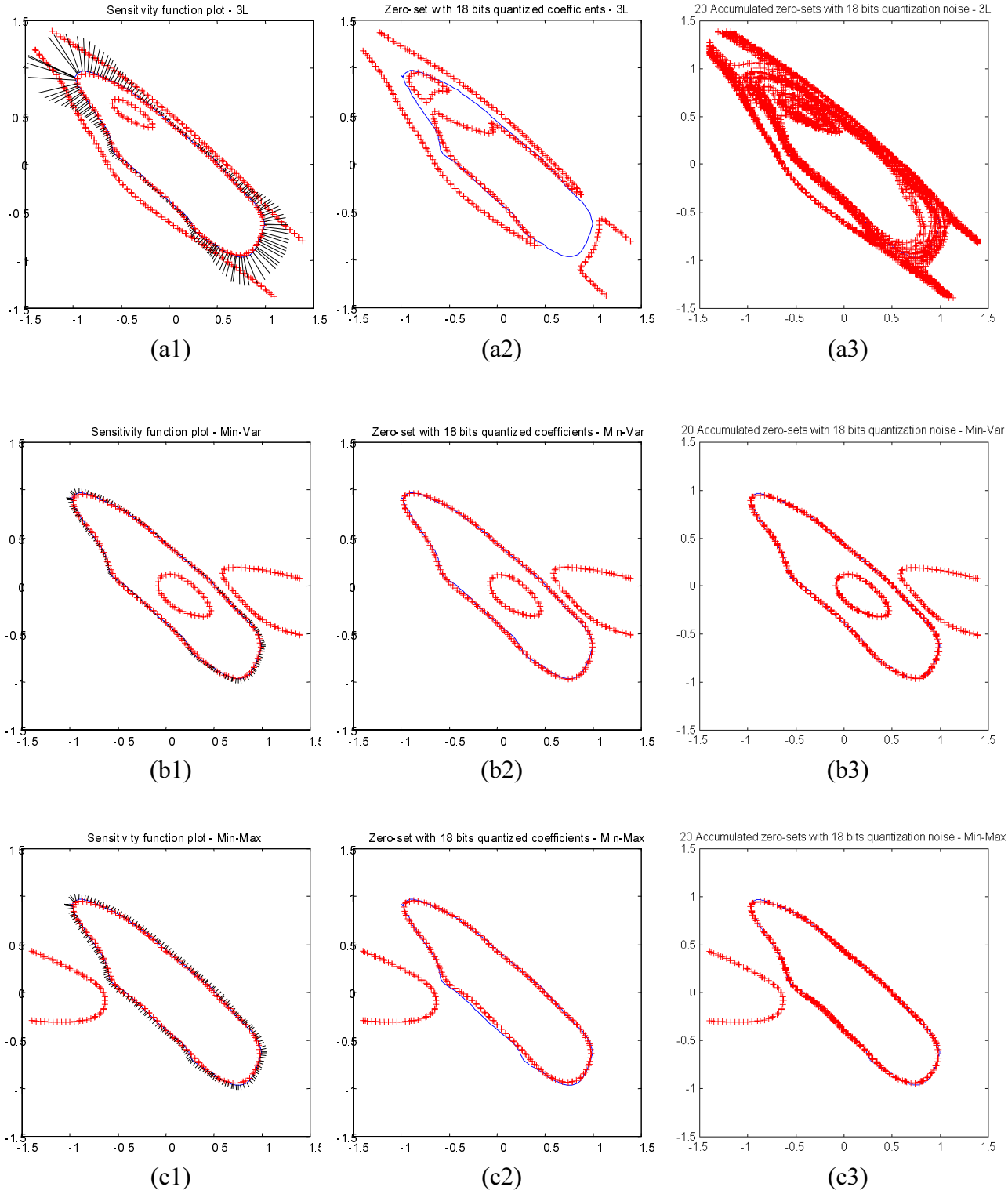


Fig. 3.6: Sensitivity and resulting errors of polynomial zero-sets using *3L, Min-Var, Min-Max* fitting.
Left column – Zero-set and sensitivity function results of polynomial fitting with un-quantized coefficients.
Center column – Zero-set after coefficient quantization to 18 bits.
Right column – 20 accumulated zero-sets of polynomials with noise added to coefficients.
Noise has independent uniform distribution with variance of 1.1e-6 (18 bits).
(a) Row – *3L*, (b) Row – *Min-Var*, (c) Row – *Min-Max*

The values of the sensitivity function using the different fitting algorithms are best compared using the histogram of the values of the function, as shown in Fig. 3.7.

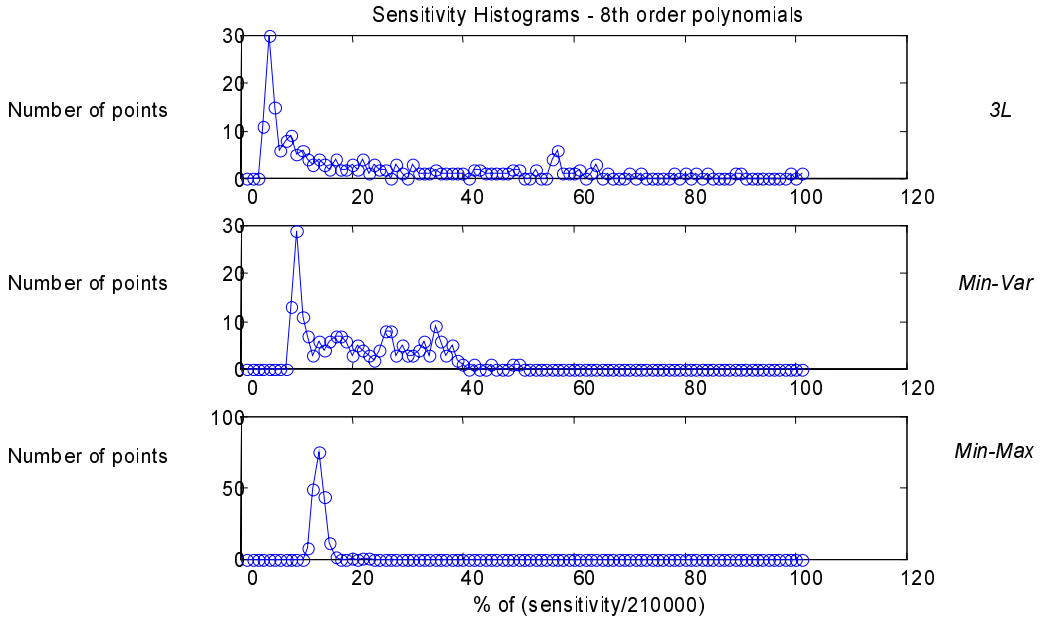


Fig. 3.7: Histogram of the sensitivity function of 8th degree polynomials using the *3L*, *Min-Var* and *Min-Max* algorithms

Fig. 3.7 shows that the sensitivity function obtained using the *Min-Max* algorithm has the most narrow histogram among the tested algorithms. Furthermore, we can see that the maximal value of the sensitivity function for all the data-points is the lowest when using the *Min-Max* algorithm – the histogram is zero for values greater than 20%, whereas the histogram ends at about 50% for the *Min-Var* algorithm and 100% for the *3L* algorithm.

From these results it is predictable that the *Min-Max* algorithm would produce the smallest fitting errors as result of coefficient quantization. Fig. 3.6(a2, b2, c3) demonstrates the zero-sets of the polynomials fitted using the *3L*, *Min-Var* and *Min-Max* algorithms with the addition of uniform distributed noise (with variance of 0.4e-6), equivalent to coefficients quantization. This figure shows the overlay of 10 plots of zero-sets obtained using different noise values.

As expected, the zero-sets of the polynomials obtained using the *3L* and *Min-Var* algorithms are less accurate than the zero-set of the polynomial obtained using the *Min-Max* algorithm. However, it is interesting to note that the location where the zero-sets of the first two figures break up is the location where the values of the sensitivity function in Fig. 3.6(a1, b1) intersect the location of spurious zero-set

points. This is a result of changes in the location of the zero-set which combine the two parts of the zero-set (one part generated due to data and one part spurious) and generate a broken zero-set.

Also notable are two properties achieved by the two fitting algorithms – *Min-Var* and *Min-Max*.

The *Min-Var* aims at a constant variance of the errors due to different values of the noise vector added to the coefficients vector. In Fig. 3.6(b2) we observe that the width of the resulting zero-set, along the original data points is approximately constant, whereas the width of the zero sets in Fig. 3.6(a2, c2) vary significantly for different data points. This is a result of the criterion that was used during the fitting of the different algorithms. The *Min-Var* algorithm finds a polynomial that generates a zero-set with constant variance, hence leading to near constant width of the accumulated zero-set plots.

More easily noticed is the constant value of the sensitivity function seen in Fig. 3.6(c1). Again, this is achieved because the criterion used by the *Min-Max* algorithm aims at finding a polynomial with constant maximal sensitivity.

To complete the discussion on the sensitivity function we present simulation results of polynomial fits obtained using 4^{th} degree polynomials using the *3L*, *Min-Var* and *Min-Max* fitting algorithms (Fig. 3.8) and also results obtained using 4^{th} order polynomials using the *non-linear* fitting algorithm (Fig. 3.9). The observations made for 8^{th} order polynomials are also valid for 4^{th} order polynomials as seen in Fig. 3.8.

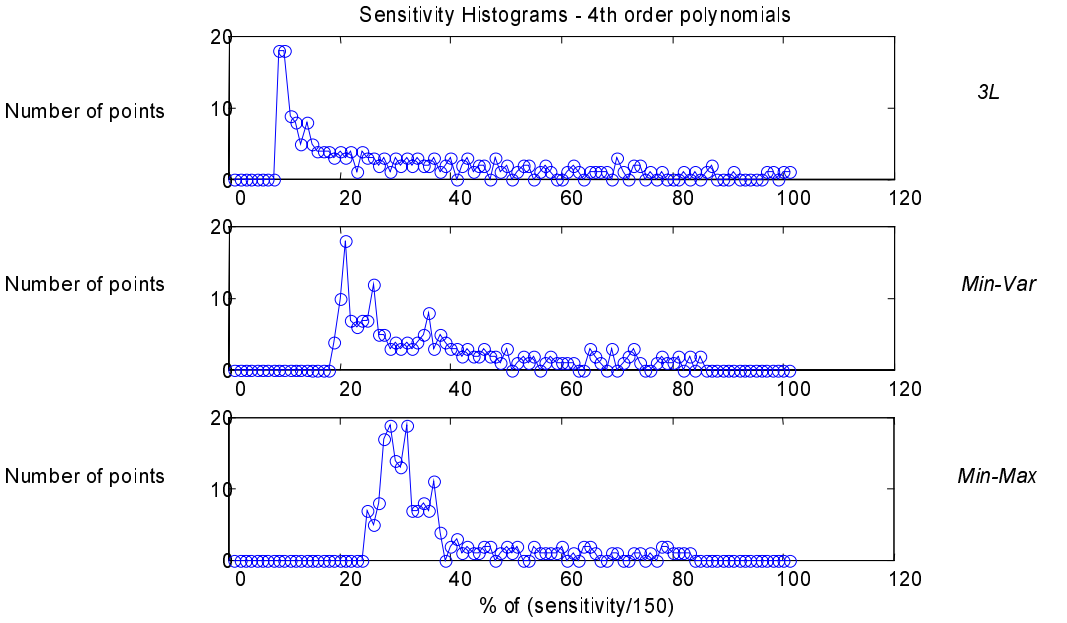


Fig. 3.8: Histogram of the sensitivity function of 4th degree polynomials using the *3L*, *Min-Var* and *Min-Max* algorithms

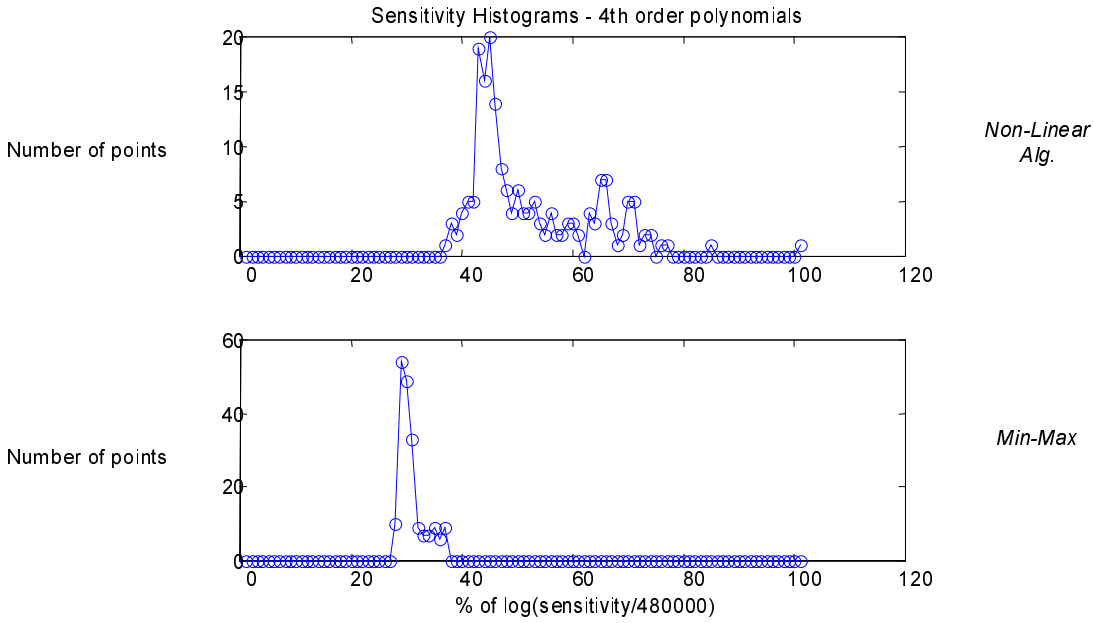


Fig. 3.9: Histogram of the sensitivity function of 4th degree polynomials using the *non-linear* and *Min-Max* algorithms

3.5.3 3D fitting

This section presents fitting results in 3D.

The fitting algorithm used here is an extension to the 2D fitting algorithm as described in section 3.4.3.

8th Order 3D polynomials

Fig. 3.10 shows two faces of a single object (left and right columns).

The top row displays the original object.

The second, third and bottom rows display fitting results using the three fitting algorithms: *3L, Min-Var, Min-Max*.

We can clearly notice the improvements in the fitting results going from the second row (*3L*) through the third row (*Min-Var*) to the bottom row (*Min-Max)*.

The improved fitting is also evident in the root mean squared-error as summarized in Table II:

Table II: RMS error obtained using 3D *3L*, *Min-Var* and *Min-Max* fitting

Algorithm =>	*3L*	*Min-Var*	***Min-Max***
σ_E	3.67	3.5	**1.91**

Where $\sigma_E = \sqrt{\frac{1}{N}\sum_{n=1}^{N} d^2(n)}$, and $d(n)$ is the distance between data point n and the zero set.

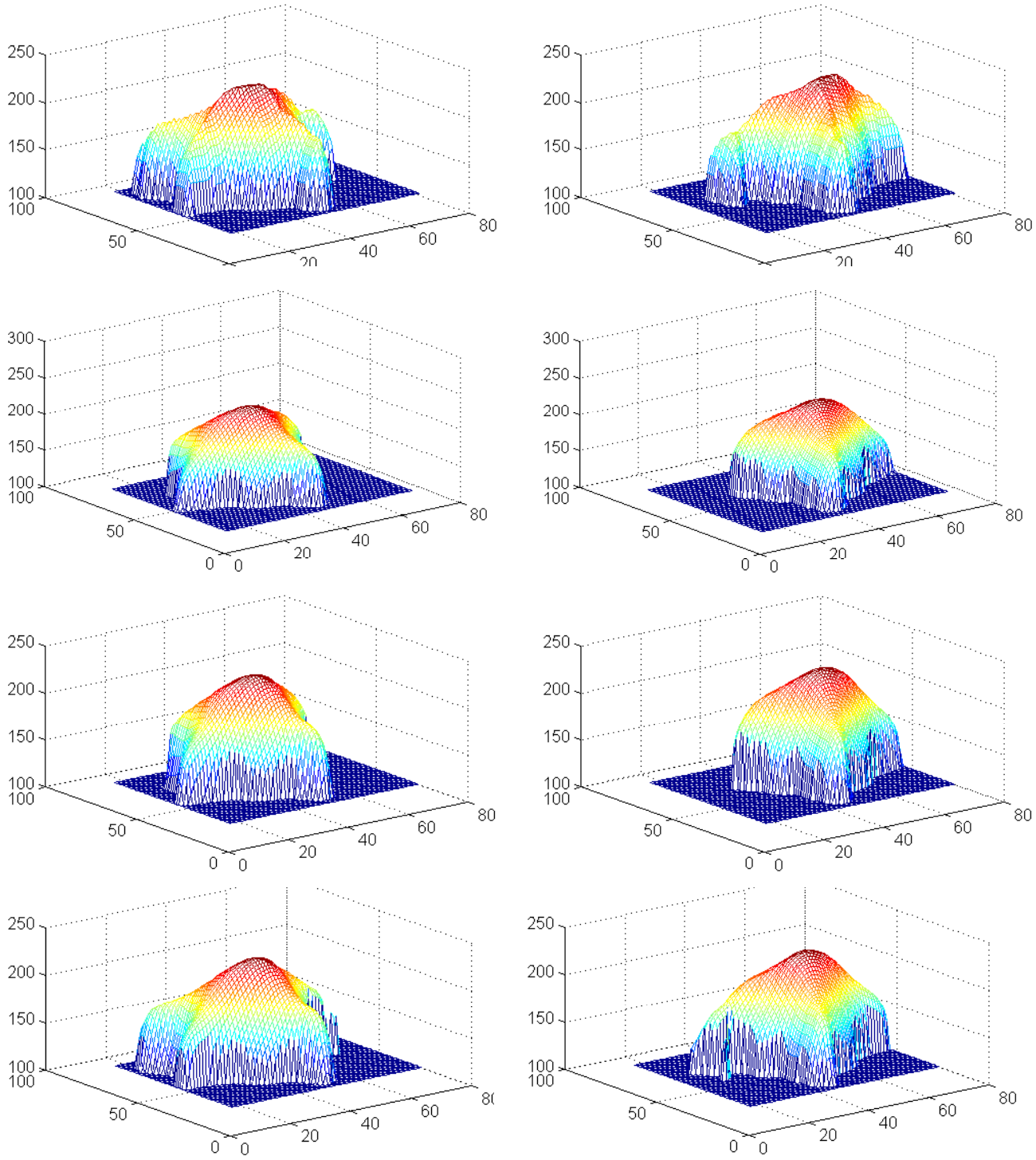


Fig. 3.10: Fitting results: Left – Shape from front, Right – Shape from back
Top row – Original data
Second row – fitting results with *3L* algorithm,
Third row – fitting results with *Min-Var* algorithm,
Bottom row – fitting results with *Min-Max* algorithm

3.6 Conclusions

In this chapter we addressed polynomial fitting to object boundaries.

The fitting algorithms presented in section 2.2.2 and in this chapter attempt to find a polynomial with the following properties:

- Zero at the location of the data points
- The direction of the gradient is perpendicular to the direction of the data points[3].
- The magnitude of the gradient is algorithm-dependent.

It is concluded that the first two requirements are trivial. To obtain a tight fit, the polynomial should be zero at the data points, and therefore, the second property follows. Therefore, the differences between different fitting algorithms lies mostly in the required magnitude of the gradients, at each data point. The fitting algorithms presented in this chapter attempt to obtain a polynomial, which is robust to quantization. The difference between the two versions of the algorithm described in this chapter – *Min-Max* and *Min-Var* – is in the type of the zero-set error that is minimized.

As seen in section 3.5, the simulations made verify that the *Min-Var* and *Min-Max* fitting algorithms described in section 3.4 yield more stable results than the *3L* algorithm, with the *Min-Max* algorithm producing better results for handling quantization effects. Furthermore, the simulations verify that it is possible to control the desired optimized characteristic by specifying a cost function which minimizes the sensitivity of that characteristic (we have attempted to minimize the maximal error and the RMS error).

Other parts of the simulation section describe fitting results of polynomials without coefficient errors. As expected, the fitting is best when using the *Min-Max* algorithm. Also, it is seen that a polynomial obtained using either fitting method includes spurious zero-set points and cannot be used for coding applications. This problem is handled in Chapter 5.

---

[3] The direction of the data points was defined as the direction of a line approximating the data points around each point.

This page intentionally left blank

Chapter 4 Curve segmentation

This chapter deals with the representation of large contours using several polynomials, each fitted to a different, non-overlapping, segment of the contour. The motivation for this approach is to save bits in the representation of the boundary, while keeping the distortion bounded. The development presented in this chapter is therefore mostly useful for coding applications.

2D implicit polynomials can represent 2D curves. Any given curve can be represented by a single implicit polynomial. However, complex curves may require a high degree polynomial, depending on the complexity of the curve and the required fitting accuracy.

For image compression applications it is desired to represent curves with a minimal number of bits. It is possible to segment a complex curve into several simple curves and achieve an overall more compact representation. The number of coefficients, the number of bits per coefficient[4] and the additional side information determine the total number of bits required for describing a polynomial. These elements are described in Table III.

Table III: Number of bits required to represents polynomials of order 1 to 8

d	#COEF	#BITs/Coef	#BITs-SI	#BITs/polynomial
Order	**Number of coefficients**	**Bits per coefficient**	**Bits in side information**	**Total number of bits**
1	3	5	20	35
2	6	6	20	56
3	10	8	20	100
4	15	10	20	170
5	21	12	20	272
6	28	14	20	412
7	36	16	20	596
8	45	18	20	830

$\#COEF = (d+1)(d+2)/2$

#BITs/Coef is a typical number

$\#BITs_{SI}$ is calculated in the previous section

the total number of bits is $\#COEF \cdot \#BITS/Coef + \#BITs_{SI}$

---

[4] The number of bits required for each coefficient used in Table III is a typical number. This number was obtained from experimental results, and greatly depends on the allowed error. Table III contains the typical number of bits for an error of 2 pixels out of 512.

We begin this section with an explanation of some necessary data normalization and calculation of the size of the required side information used in the segmentation process (sections 4.1). In section 4.3 we present two contour segmentation algorithms. The high complexity of an optimal exhaustive search, calculated in section 4.3.1, motivates the development of top-to-bottom and bottom-to-top sub-optimal algorithms described in sections 4.3.3 and 4.3.4, respectively.

4.1 Data normalization

The first necessary step is data normalization, e.g., the location of the object is normalized in order to achieve a compact representation. We apply the following transformation to each of the data-set points:

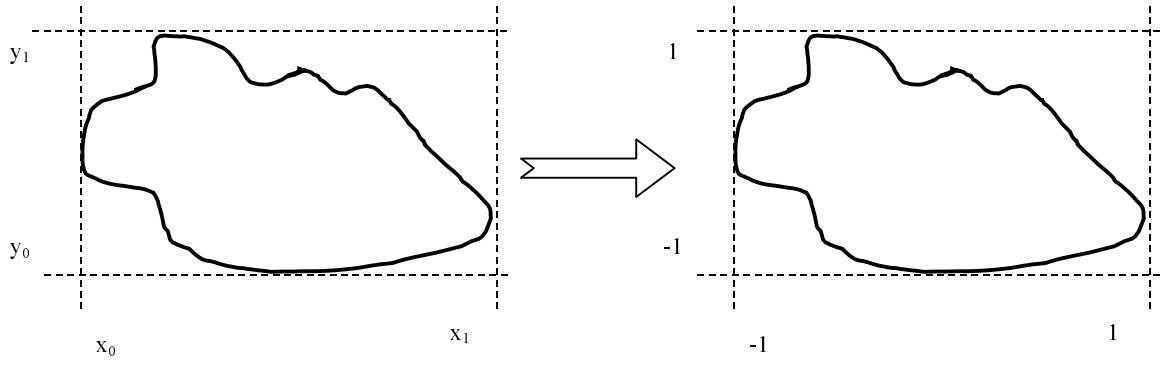


Fig.4.1: Coordinate transformation for data normalization
(a) - Original data location $p_n = (x_n, y_n)$, $n = 1,..., N$
(b) - Normalized data location $\widetilde{p}_n = (\widetilde{x}_n, \widetilde{y}_n)$, $n = 1,..., N$

This transformation can be performed using:

$$\widetilde{p}_n = g(p_n - c) \tag{4.1}$$

where c is the offset vector and $g = \begin{pmatrix} g_X & 0 \\ 0 & g_Y \end{pmatrix}$ is a gain matrix. The monomial vector contains exponents of the x and y locations of the data points. We wish to limit the possible values of the monomial components. By normalizing the data to location between -1 to 1, we limit the possible values of the monomial to -1 to 1 as well. Large values of the x and y coordinates would cause high ratios between the values of the leading form and the low order monomials, resulting in high ratios between the resulting polynomial coefficients. This in turn would cause numerical difficulties during the calculation of the polynomials, and later would require more bits for the representation of the coefficients in fixed point.

Therefore, we normalize the data to lie between -1 and 1 by setting:

$$\begin{aligned} x_0 &= \min(x_n), x_1 = \max(x_n) \\ y_0 &= \min(y_n), y_1 = \max(y_n) \end{aligned} \tag{4.2}$$

and the gain and offset values are calculated according to:

$$\left.\begin{aligned} (x_1 - c_X)g_X &= 1 \\ (x_0 - c_X)g_X &= -1 \end{aligned}\right\} \quad \Rightarrow \quad c_X = \frac{x_0 + x_1}{2}, g_X = \frac{2}{x_1 - x_0}$$

$$\left.\begin{aligned} (y_1 - c_Y)g_Y &= 1 \\ (y_0 - c_Y)g_Y &= -1 \end{aligned}\right\} \quad \Rightarrow \quad c_Y = \frac{y_0 + y_1}{2}, g_Y = \frac{2}{y_1 - y_0} \tag{4.3}$$

4.2 Curve segmentation information

When describing a curve using several implicit polynomials, some side information is required. By side information we denote the additional information needed for the reconstruction of the contour in addition to the polynomial coefficients.

In order to merge the zero-sets of several polynomials, describing neighboring segments a starting point of each segment must be included in the side information (see Chapter 5). The different segments of the curves may be fitted using polynomials of different orders. Therefore, the order of the polynomial should also be included in the side information.

Combining both normalization information, starting point and polynomial-order for each segment, we obtain $\#BIT_{SI}$ - the necessary amount (in bits) of side information required for coding:

$$\#BITs_{SI} = 2(\log_2 S_X + \log_2 S_Y) + N_{SEG}(\log_2 S_X + \log_2 S_Y + \log_2(MAXORDER)) \tag{4.4}$$

where S_X and S_Y are the horizontal and vertical sizes of the image respectively, N_{SEG} is the number of segments and $MAXORDER$ is the maximal allowed order of the polynomial. We also need to allocate a maximum[5] of $\log_2 S_X + \log_2 S_Y$ for the starting point of the scan of each polynomial. For the entire contour, instead of using an offset and gain to code c, g (from the previous section), we use x_0, x_1, y_0, y_1 which can be represented using $2(\log_2 S_X + \log_2 S_Y)$.

[5] The number of bits required for the starting point is $\log_2 g_X + \log_2 g_Y$, and is bounded by $\log_2 S_X + \log_2 S_Y$.

Therefore, for a 512x512 pixels image, using up to 4'th order polynomial we obtain:

$$\#BITs_{SI} = 2(\log_2 512 + \log_2 512) + N_{SEG}(\log_2 512 + \log_2 512 + \log_2 4) = 2 \cdot 18 + N_{SEG} \cdot 20$$

The amount side information has a constant value of 36 bits, used for normalization purposes and 20 bits per segment. Since the constant 36 bits are always required when using IPs to describe a boundary, we only count the additional 20 bits per segment in our comparison in Table III.

4.3 Curve segmentation algorithms

The segmentation algorithm is a critical stage in the process described in the previous sub-section. The performance of this algorithm determines the performance of the entire compression system.

When looking for a segmentation algorithm we need to define a cost function (or criterion). This cost function should include the rate, the distortion and whatever other parameters we are interested in (for example complexity). The distortion itself should also be defined, since many metrics for distortion measurement are possible.

4.3.1 Exhaustive search for optimal curve segmentation

For a given curve, and a given cost function, we expect that there exists a segmentation that minimizes the cost function. This segmentation can be found by examining the costs of all possible segmentations and choosing the one with a minimal cost. This process may be extremely complex, since the number of possible segmentations for even a short curve is very high.

Each two data points on the curve can either belong to the same segment or belong to two different segments. We can look at the segmentation in a graphical way as an equivalent chain of elements, where each two elements may either be connected or unconnected. Connected elements represent data points that belong to the same segment. Unconnected elements represent data points that belong to different segments. For a closed curve, this equivalent chain has exactly as many elements as the number of data points. For an open curve, the equivalent chain has a number of elements equal to the number of data points minus one.

For closed curves, the number of possible segmentations is therefore 2^{Np} where Np is the number of points on the curve. However, in addition, each segment may

also be represented by a polynomial of a different order. A calculation of the number of combinations required for an exhaustive search is presented in Appendix B.

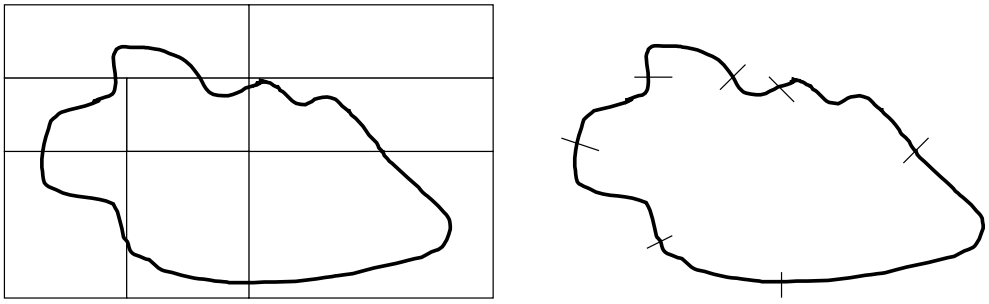
In order to implement an exhaustive search, different order polynomials should be fit to every possible segment, and the resulting cost tested.

It may be possible to eliminate many of the segmentations during an exhaustive search, and speed the process (for example, by using a segmentation tree, and dismissing possible segmentations for which and accumulated cost of the root is greater than the best cost encountered). However, this elimination would have to be very drastic in order to reduce the complexity to a tolerable level.

Since an exhaustive search for the optimal segmentation is impractical, a sub-optimal algorithm (of reasonable complexity and performance) is needed. In this section we examine two segmentation algorithms. The first is a *top-to-bottom split-and-merge* algorithm, the second one is a *bottom-to-top merge* algorithm. We describe these algorithms and investigate their properties in the following subsections.

4.3.2 Properties of 2D curves

For segmentation purposes, 2D curves can be considered as either 2D images or as a 1D array of point coordinates. The difference between the two approaches can be seen in Fig. 4.2.



(a) (b)

Fig. 4.2: Two approaches to curve segmentation
(a) – Curve segmented as a 2D image
(b) – Curve segmented as a 1D array of points.

Since we use implicit polynomials to describe the segments, we would like to preserve the continuity of segments. The 2D representation (Fig.(a)) does not conserve the sequence of the points making up a curve, while the 1D representation (Fig.(b)) inherently maintains the order of the points.

Although it is possible to derive segmentation algorithms, for both 2D and 1D representations, we prefer to use the 1D representation and maintain the order of the points for segmentation, polynomial fitting and reconstruction.

We now distinguish between two types of curves – open and close. Open curves have starting and ending points. Closed curves are cyclic. When segmenting an open curve, the first point of the first segment, and the last point of the last segment are always the first and last points on the curve. When segmenting closed curves, there is no defined starting and ending points and no first and last points on the curve.

4.3.3 Top-to-bottom curve segmentation algorithm

Top-to-bottom segmentation is very popular in 2D image compression. An image is recursively split into regions with equal proportions as long as a cost function is being reduced by these splits. Fig. 4.3 shows an example of a 2D quad-tree segmentation of an image.

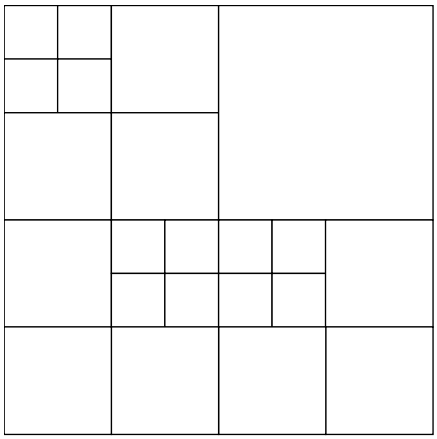


Fig. 4.3: Segmentation of a 2D image

For image processing this segmentation algorithm has several appealing properties:

- Low complexity (fast).
- Efficient representation of segmentation results, e.g. using quad-trees [22].

Since the algorithm described above was originally developed for 2D images, some adaptation is required for the 1D curves we are dealing with here. The flowchart of the original algorithm remains, but the split and merge operations are now performed in 1D, resulting in split and merge of lines instead of regions.

Description of top-to-bottom curve segmentation algorithm

Throughout the algorithm implicit polynomials are being fitted to curves. For each curve a polynomial with the lowest possible order is fitted, so that a maximal allowed distortion value is not exceeded. To implement this, a scan is performed on the order of the polynomial, starting from 1 and going up to the maximal order that is allowed. When the resulting distortion is below the maximal value allowed, the scan is terminated with a successful fitting result. When the maximal allowed order of the polynomial is exceeded, the scan is terminated without a valid polynomial fitting result.

The presented top-to-bottom segmentation algorithm consists of two steps – initialization and optimization (see Fig. 4.4).

The initialization step begins with a single segment containing the entire boundary. This segment is examined for maximal fitting error. If the fitting error is above the maximal allowed value, the segment is split into N_{SPLIT} equal parts, i.e., each with the same number of points[6]. This process is recursively repeated for each resulting segment until all segments are fitted by implicit polynomials with an error that is below the maximal allowed. The order of the IP representing each segment is set to the lowest possible order so that the distortion is less than the maximal allowed.

Once the initialization step is complete, all the segments are represented by IPs with a distortion that is below the limit, and the optimization step begins. In this step, split-and-merge actions are attempted, with the goal of lowering the value of a cost function. The cost function indicates the overall quality of the segmentation in terms of rate and distortion. To accomplish this, a split is attempted on each segment, and a merge is attempted on each two neighboring segments. The resulting value of the cost function from each of these segmentation attempts is calculated, and the segmentation that produced the lowest cost is selected. The algorithm continues until the segmentation remains constant.

[6] In our simulations we used $N_{SPLIT} = 3$

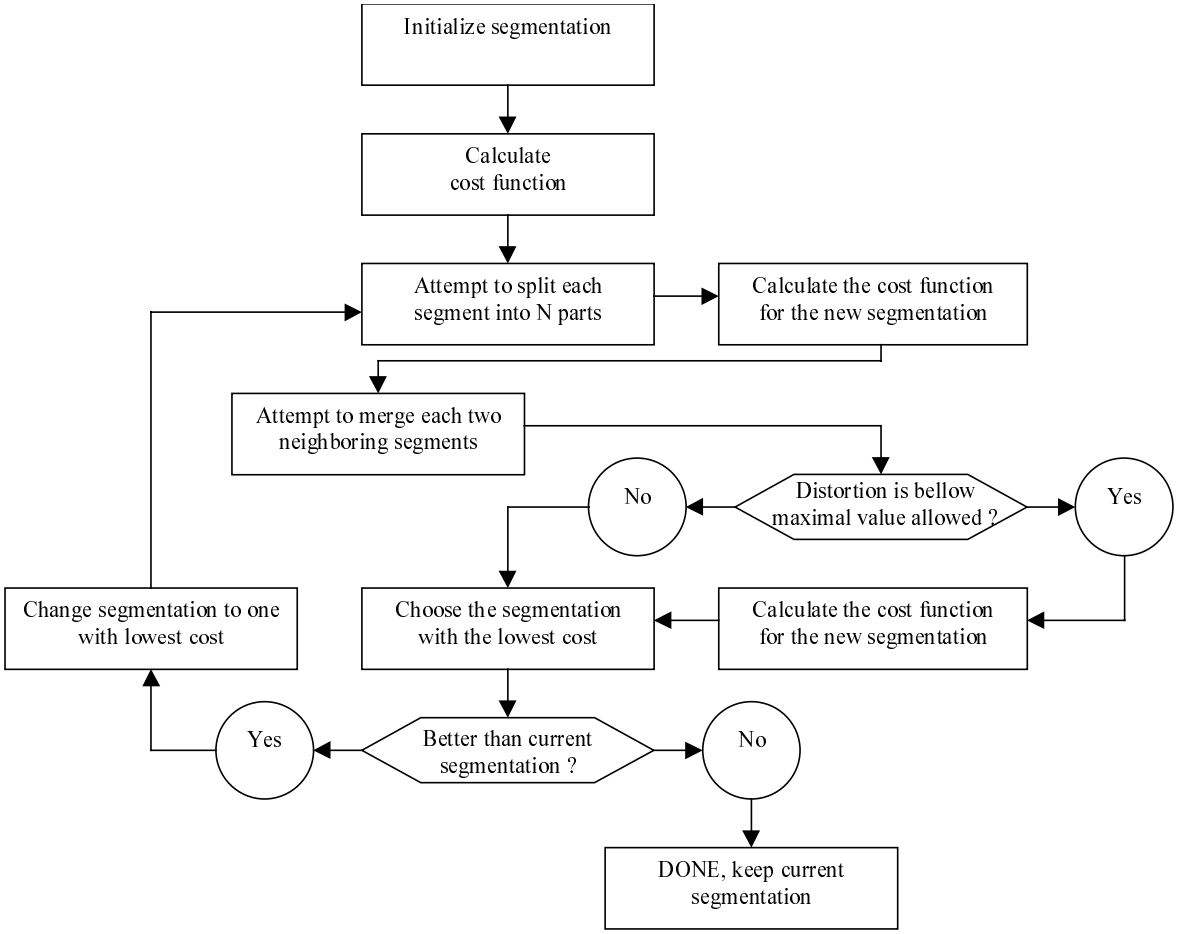


Fig. 4.4: Flow chart of top-to-bottom segmentation

The initialization process is required for a successful optimization because of the fitting properties of implicit polynomials. When arbitrarily splitting a segment that has been poorly represented by an IP, poor fitting results may still be obtained for each of the new segments. That is, only a small reduction in the distortion of each of the segments after the split is obtained, while the rate actually increases (because after the split more polynomials need to be coded). It is possible however that good fitting could be achieved by splitting at other points, but the top-to-bottom segmentation algorithm is based on an a-priori splitting scheme. Poor fitting results encountered after the first split may cause the algorithm to terminate (with the complete contour as one segment), not having the opportunity to explore other splitting points.

The value of the cost function may provide reliable information about the overall quality of a fit to a contour when the fitting is good. However, when the fitting is poor, the value of the cost function may be misleading, and making segmentation decisions based on that information may be wrong. Therefore, beginning the optimization with a segmentation, for which all the segments are properly fitted (with

an error that is less than the maximal), allows the evaluation of the quality of subsequent merges of splits using the cost function, without leading to wrong conclusions.

In order to obtain the rate and distortion of a given segmentation, IPs should be fitted to each segment. When the segmentation changes there is no way to calculate the expected rate and distortion of the new segmentation, based only on the rate and distortion of the original segmentation (rate depends on the required order and distortion depends on the fitting error). Therefore, new IPs should be fit to the segments that where changed, and the rate and distortion should be re-calculated for these segments. In order make optimal split or merge decisions, we must explore all options for splits and merges. This approach however, has extremely high complexity and is not practical. We would have to examine $\prod_{i=1}^{n_{SEG}} size(i)$ splits (where $size(i)$ is the size of segment i and n_{SEG} is the number of segments) and $2^{n_{SEG}}$ merges (see 4.3.1). Since the number of polynomial fits required to test all possible options is large, we only test part of the possible segmentations, in the hope that a good segmentation would ultimately be achieved.

In this work, we have examined all possibilities of N part splits (splitting each segment into N equally partitioned parts) and the merging of each two neighboring segments. This process is shown in the block diagram in Fig. 4.4. This results in n splits n merges and an overall of n $(1+N)$ polynomial fits required to decide on the best next segmentation given the current segmentation.

If the cost obtained by one split or one merge action (as defined above) is lower than the cost of the current segmentation, than, the action leading to that cost is taken. If not, the algorithm terminates with the current segmentation.

Cost function for top-to-bottom curve segmentation

Selection of a cost function for the algorithm is critical, and determines the performance of the algorithm. When running the algorithm with two different cost functions we could expect to obtain two different segmentations.

Notation

n_{SEG} - Number of segments.

R_n - Rate for segment n. This is the total number of bits required for the representation of segment n, and is only dependent on the order of the polynomial used to represent the segment.

D_n - Distortion of segment n. In this work the distortion has been defined as the maximal distance between a data point to the nearest zero of the polynomial.

$\#_n$ - Number of points in segment n.

$R = \sum_{n=1}^{n_{SEG}} R_n$ - Total rate needed to represent the curve.

$D = \max\{D_n\}$ - Total distortion between the fitted polynomials and the data.

C - Cost function. The goal of the optimization is to minimize this function.

Various cost functions

In our simulations, several cost functions were examined.

The first function evaluated is:

$$C_1 = RD \tag{4.5}$$

The obvious goal of this function is to decrease the total number of required bits and the distortion. The function C_1 has no consideration for the number of data points in the segment with the maximal distortion. This cost function considers only the maximal distortion among all segments. Because changes in the segmentation – leading to an improved fitting (lower local distortion) do not affect the overall distortion and the cost, this criterion makes it difficult for an iterative algorithm to find the global optimum (lowest cost).

The second cost function evaluated is:

$$C_2 = \sum_{n=1}^{n_{SEG}} \#_n \, R_n D_n \tag{4.6}$$

This cost function overcomes the problem encountered with C_1 because each segment contributes its proportional part to the distortion according to the number of points represented.

This approach, however, raises stability problems. Large segments, which usually require a polynomial of high order dominate the cost of the segmentation. Small

segments, that can usually be represented using a low order polynomial have a very small contribution to the cost function. The algorithm will therefore prefer splitting large segments over merging small segments that can be combined into larger segments, even with a good fit.

The third cost function evaluated is:

$$C_3 = \sum_{n=1}^{n_{SEG}} \frac{R_n D_n}{\#_n} \tag{4.7}$$

The rational behind this function is that large segments should be encouraged. The function actually calculates the sum of rate-distortion product per data point, for each segment. This cost function balances between the contribution of small and large segments. Every point in the entire curve actually contributes its effective local cost. Therefore, the algorithm strives to reach a segmentation that achieves a uniform cost for each point on the curve. Since there is no given priority for the fitting of one point over the other[7], there is no reason to give priority to the cost of the representation of one point over the other.

When the distortion is fixed to the maximal allowed distortion this function is reduced to:

$$C_3\left(D_n = D_{MAX}\right) = D_{MAX} \sum_{n=1}^{n_{SEG}} \frac{R_n}{\#_n} \tag{4.8}$$

The algorithm would than prefer segments with a high ratio between the number of data points and the number of bits required for representation (see Table III). Since the algorithm forces splits on segments with larger distortion than allowed, a constant distortion is often reached.

The fourth and last cost function evaluated is:

$$C_4 = \sum_{n=1}^{n_{SEG}} \frac{R_n + \lambda D_n}{\#_n} \tag{4.9}$$

This cost function is often used [23,24] as a criteria for rate-distortion based segmentation algorithms. The parameter λ is selected to balance between tight fit and low bit rate. Accordingly, the parameter λ is set to the ratio between the expected typical rate and the allowed distortion.

---

[7] Giving priority to some points over others could be implemented either through the cost function used by the segmentation, or directly via the distance metric.

Using this cost function, we obtained results similar to those obtained using C_3.

Among the different cost functions presented here, the function C_3 produced the best segmentation results and was used for our simulations.

Advantages of top-to-bottom curve segmentation algorithm

The top-to-bottom segmentation algorithm described above shows convergence to a fixed point, as seen in section 4.4.1.

Because the split and merge actions take into account the actual error in the fitting of the implicit polynomial, the distortion measure gives a good indication of what action (split or merge) is optimal at each stage.

The algorithm does not require any manual initialization. No "special" points are required from an operator, and the fitting quality of the implicit polynomial is what determines the resulting segmentation.

Because the algorithm attempts both split and merge at any stage, segments may be formed and then split according to a global cost function. There is no constraint on performing only splits or merges, and the algorithm may, depending on the type of cost function and boundary data, advance towards a global optimum by bypassing local peaks.

Disadvantages of the top-to-bottom curve segmentation algorithm

The primary disadvantage of the top-to-bottom segmentation algorithm presented above is the complexity involved in segmenting an object. Because of the nature of implicit polynomials, good potential splits are found at very specific points along curves. E.g., a lower cost is usually achieved when segmenting a curve at some special points while arbitrary segmentation of a curve is much more costly. However, because these points are not necessarily easily recognizable by heuristic algorithms (such as high curvature detection algorithms), many split actions into small segments may be performed before merges are efficient (when considering the cost function). This problem is fundamental for top-to-bottom segmentation, and is also the reason why different cost functions produce very different segmentation results. For example, consider the object in Fig. 4.5:

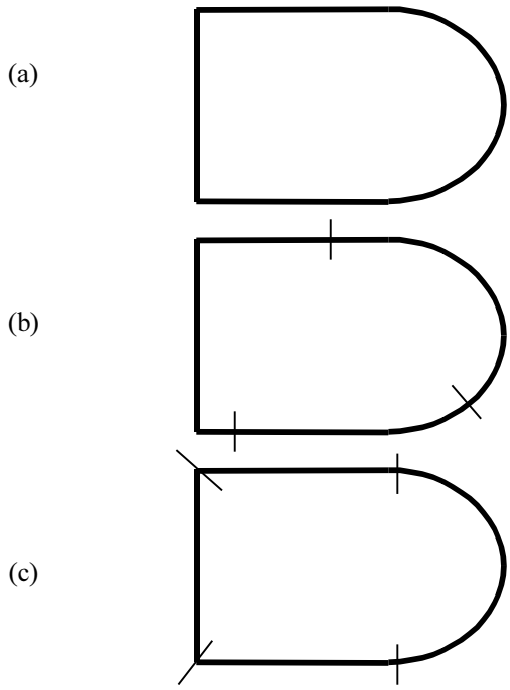


Fig. 4.5: Example object for segmentation
(a) – un-segmented, (b) – default 3 parts segmentation, (c) – optimal segmentation.
Boundary is plotted in solid line, segmentation location is marked by thin lines

The top-to-bottom segmentation algorithm begins with the un-segmented curve in Fig. 4.5(a). The first decision made is to split it. This results in the segmentation presented in Fig. 4.5(b). The final segmentation should be as presented in Fig. 4.5(c). Because the segmentation points in Fig. 4.5(c) are not found on the split grid of the segments in Fig. 4.5(b), further splits are required before merges can be performed. The algorithm easily senses that more splits are required because these splits reduce the fitting error (distortion) and allow usage of lower order polynomials (reduced rate). When a sharp edge is found in a body (such as in the upper right corner of the body), a very high degree polynomial is required to fit both sides of the corner. However, two low order polynomials can accurately describe that corner. The top-to-bottom algorithm yields the desired result, but in order to achieve it, a split into very small segments is first made (as dictated by the cost function). This property of the solution (which is common to many shapes) serves as a motivation for the bottom-to-top segmentation algorithm presented in the next section.

4.3.4 Bottom-to-top segmentation

The top-to-bottom segmentation algorithm presented in the previous section has several drawbacks that motivated the development of the bottom-to-top algorithm presented in this section. Observations made during execution of the top-to-bottom algorithm indicate that, at first, the contour is split into many small curve segments, and later these segments are merged into larger ones. Convergence is obtained when there are no more segments to merge. This behavior is a direct result of the fitting properties of implicit polynomial (as described above). Because of this behavior of the top-to-bottom algorithm, it seemed sensible to begin segmentation with minimal size segments and to perform segment merges only.

Bottom-to-top segmentation is a merge algorithm that begins with minimal size elements comprising a given boundary and attempts to merge neighboring segments in order to reduce a specified cost function (typically combining rate and distortion). The minimal size of the elements is the number of points in a segment to which a polynomial of the initial order may be fitted. For 1^{st} order polynomials, the minimal size of the elements is two points.

Since this algorithm considers only segment merges, the bottom-to-top algorithm is more systematic and is faster than the top-to-bottom algorithm.

Description of bottom-to-top segmentation algorithm

We consider the boundary as a 1D array of points.

As initialization for the merge process, minimal size segments are formed from the set of data points (see Fig. 4.6).

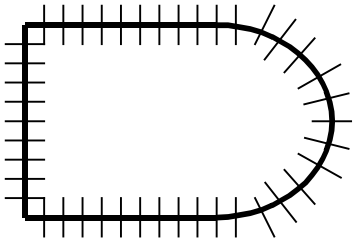


Fig. 4.6: Object segmented into minimal sized segments

A maximal distortion bound is placed on each segment. Each two neighboring segments are considered for merging, and a fitting error (distortion) is calculated for each new potential segment (merge result). A sorted stack is built containing merge

candidates, with the fitting errors serving as sort indices for the stack. In each iteration the top of the stack (merge with lowest fitting error) is fetched and executed. The stack is scanned and merge candidates that use one of the merged segments are removed. A merge is attempted between the new segment that was now formed (by the merge operation) and it's neighbors, and the merge results are entered into the stack. The process continues as long as there are elements in the stack with fitting errors below the maximal error allowed.

In order to make a comparison between two segments on the basis of distortions alone, the rate of all segments has to be constant. To accomplish this, the algorithm is executed in stages, where each stage is executed on polynomials of the same order.

Initially, 1^{st} order polynomials are used. The process described above is executed until termination – i.e., when all the possible merges have been made. Then, the order of all the polynomials describing the segments is raised by one and the process is repeated, beginning with the segmentation result of the previous order. In principle, the order of the polynomial can always be increased, and the merge process can continue until there is one segment containing the entire object boundary. Practically, the maximal order of the polynomials is limited. In our work it was limited to four, achieving good segmentation results.

This process is described in Fig. 4.7.

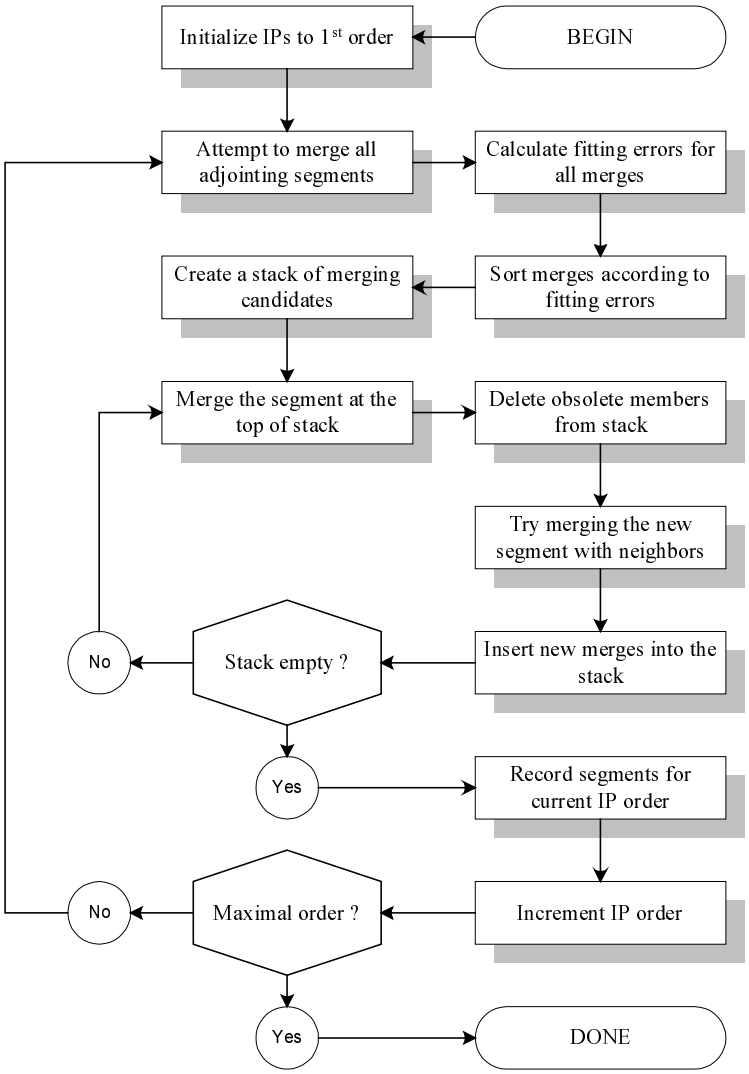


Fig. 4.7: Bottom-to-top segmentation flow chart

This algorithm does not require the specification of a cost function. However, since the goal of the algorithm is to produce a segmentation which allows efficient coding in terms of rate and distortion, we would like to define a cost function that allows us to evaluate the resulting segmentation. In this algorithm, the distortion is always kept bounded, and merges are performed as long as the maximal allowed distortion in not exceeded. We therefore regard the distortion as constant, and the cost function we evaluate is simply the rate.

The rate required for representation is constantly improving as long as merges continue on the same order. However, when increasing the order of the polynomials, the rate required rises, and the cost increases.

Finding an optimal segmentation

Because the rate decreases when merges are performed and increases when the order increases, we must find a segmentation that produces the best segmentation, among the possible segmentations encountered during the merge process. We construct a segmentation tree, including the possible segmentations encountered during the merge process. Since for each order of the polynomial, the lowest rate is obtained when no more segmentation is possible, these segmentations make up the search tree.

When the merge process is complete, a segmentation tree looks like the tree shown in Fig. 4.8. This tree describes the final segmentations for each polynomial order and the relations between them. Since the algorithm performs only merges, each parent segment is made of complete child segments, and therefore the connections between the segmentations at different polynomial orders can be described as a tree.

Each node on the tree in Fig. 4.8 describes a segment. Although each segment may describe a different number of data points, and the number of bits required for its representation depends only on the order of the polynomial. This number is listed in the right-hand column in the figure. A segment may be described by the parent or by its children (recursively).

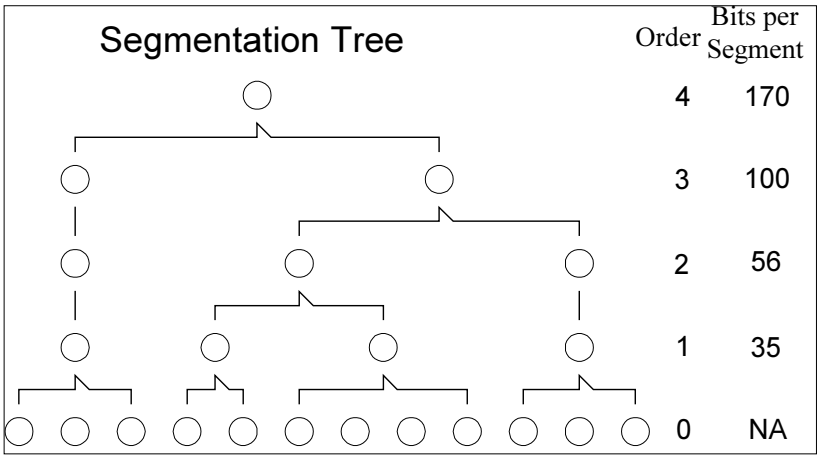


Fig. 4.8: Example of a segmentation tree

We use a search algorithm that finds the best choice of segments, resulting in the minimal number of bits for the representation of the segmented contour.

The scan algorithm is performed in two passes:

- First pass: In this pass going from bottom up, the number of bits required for the representation of each node is recorded. When moving from the child nodes to their parent, the algorithm compares between the number of bits required for representation of the parent (according to its order) and the accumulated number of bits of its children. The number of bits required for the parent is the lower of the two. This process continues until the top order node (or nodes) is reached. Relating to the example in Fig. 4.8, all 1st order nodes can be represented using 35 bits per segment. The left-most and right-most 2nd order nodes can both be represented by their child nodes. They can be represented using 56 bits as 2nd order polynomials but could be represented by 35 bits if 1st order polynomials are used. The center 2nd order node is more efficiently represented using a 56 bit 2nd order polynomial than the sum of two 35 bits 1st order polynomials. In the same manner, the 3rd and 4th order nodes are evaluated. The results of this pass are shown in Fig. 4.9.

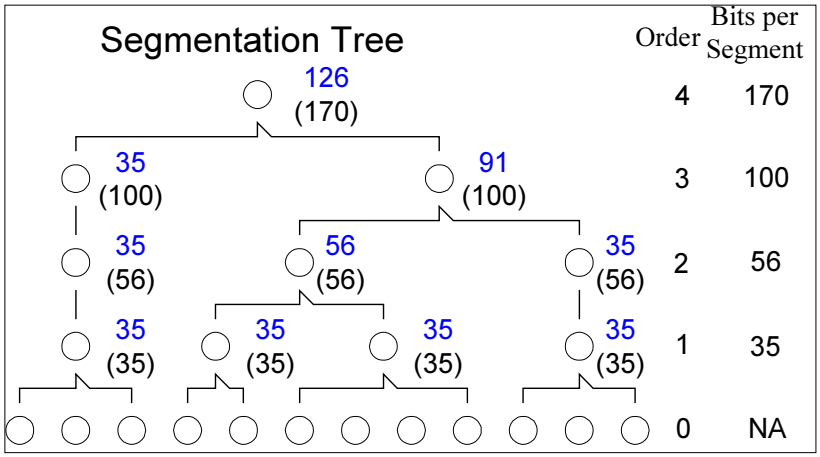


Fig. 4.9: Segmentation tree example after first pass
The minimal number of bits required to represent a node is shown next to it.
The number of bits required to represent the node by a single polynomial appears in parentheses.

- Second pass: In this pass, going from top-to-bottom, each node is tested for optimality against its siblings. The comparison is made between the minimal number of bits required to represent the node against the number of bits required to represent the node using a single polynomial according to its level (order). If the tested node is found optimal, the scan is complete

for that branch. If not, the scan is recursively continued down until stopped. Continuing the example tree in Fig. 4.9, the 4th order node is not optimal and its children are scanned. The scan on the left-hand branch stops at the 1st order node. The scan on the right-hand branch divides again at the 3rd order node, which is not an optimal node. The right-hand 1st order node and the center 2nd order node are selected as optimal nodes. The result of this pass is shown in Fig. 4.10.

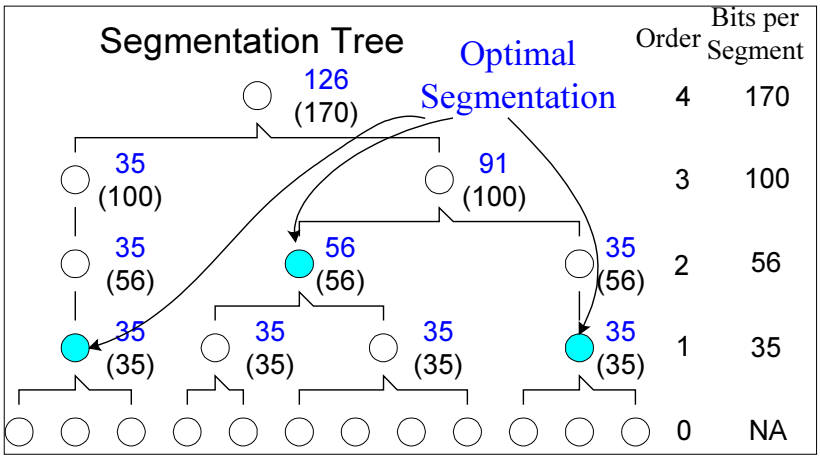


Fig. 4.10: Segmentation tree example after second pass
Nodes selected for optimal coding are marked.

Upon termination, the scan process returns the choice of nodes, whose representation requires the minimal number of bits, among all other segmentations scanned during the merge process and described by the segmentation tree.

Advantages of bottom-to-top curve segmentation

The main advantage of the bottom-to-top segmentation algorithm presented here lies in its low and predictable complexity. For each polynomial order, each segment may be fitted twice – once during the initialization phase (when a stack of fits is constructed), and second, after the segment is merged with a neighbor. Since there may be no more merges than the number of segments (n_{SEG}) per polynomial order, there may be at most $2n_{SEG}$ polynomial fits for each order. Limiting the highest order to 4 (as was done in the simulations), and beginning with initial segments of two

contour points, we obtain a maximum of $4 \cdot Np$ polynomial fits[8]. This number is significantly lower than the 2^{Np} segmentation tested in the exhaustive search. Since during each iteration of the algorithm the number of segments decreases, down to only several large segments for the 4th order polynomials.

Disadvantages of bottom-to-top curve segmentation

The main drawback of this algorithm is the fact the only merges are considered. Although the algorithm merges segments with best fitting (resulting in lowest distortion), the fitting errors of higher order polynomials for the same segments cannot be predicted. The higher degree of freedom in higher order polynomials ensures that there always exists a polynomial with a higher order whose zero-set includes the zero-set of the polynomial of lower order. Therefore, the assumption made by the algorithm is that segments which are described well by low order polynomials would also be described well by high order polynomials. However, the converse is not true – i.e., segments with poor representation using a low order polynomial may have a good fit using higher order polynomial. Therefore, making the best merge decisions for lower order polynomials doesn't assure that these decisions are optimal after the order raise.

Because we cannot predict the quality of fitting of higher order polynomials, it seems that this problem may only be completely resolved using the impractical exhaustive search.

4.4 Simulation results

Here we present simulation results of the segmentation algorithms presented in the previous section. The data used for the simulations was obtained from segmented medical images.

We display the segmentation results of the top-to-bottom and bottom-to-top segmentation algorithms.

Since an exhaustive search for the optimal segmentation is impossible to perform, it is difficult to check how close are the resulting segmentations to the optimal one.

[8] The stack may contain up to $Np/2$ elements, and is constructed 4 times (for each polynomial order). This is a loose upper bound, but sufficient for our discussion.

We do know that the optimal segmentation is invariant to rotation, i.e., a rotated curve would have the same optimal segmentation as the non-rotated curve. Therefore, we know that if we obtain two different segmentations before and after rotations than the segmentation algorithm yields a result, which is probably significantly different than the optimal segmentation. Of course, the converse cannot be assumed. We made all simulations using the original and rotated data to check this property.

4.4.1 Top-to-bottom segmentation

Three cost functions were proposed in section 4.3.3 for top-to-bottom segmentation. The first two cost functions caused severe difficulties that were also described. We have therefore concentrated in simulating the results produced by the third cost function, i.e. $C_3 = \frac{1}{\sum_{n=1}^{n_{SEG}} \frac{R_n D_n}{\#_n}}$. For this simulation we used $N_{SPLIT} = 3$, i.e., each segment is split into equal parts to evaluate the cost of segment splits. Smaller values of N_{SPLIT} result in faster calculations and larger value result in a finer resolution of splits and greater complexity. Larger values of N_{SPLIT} produced similar results (in term of cost). In order to ensure the desired properties of the cost function, we have limited the maximal fitting error to 3 pixels.

The results of segmenting two object contours, before and after rotation are presented in Fig. 4.11.

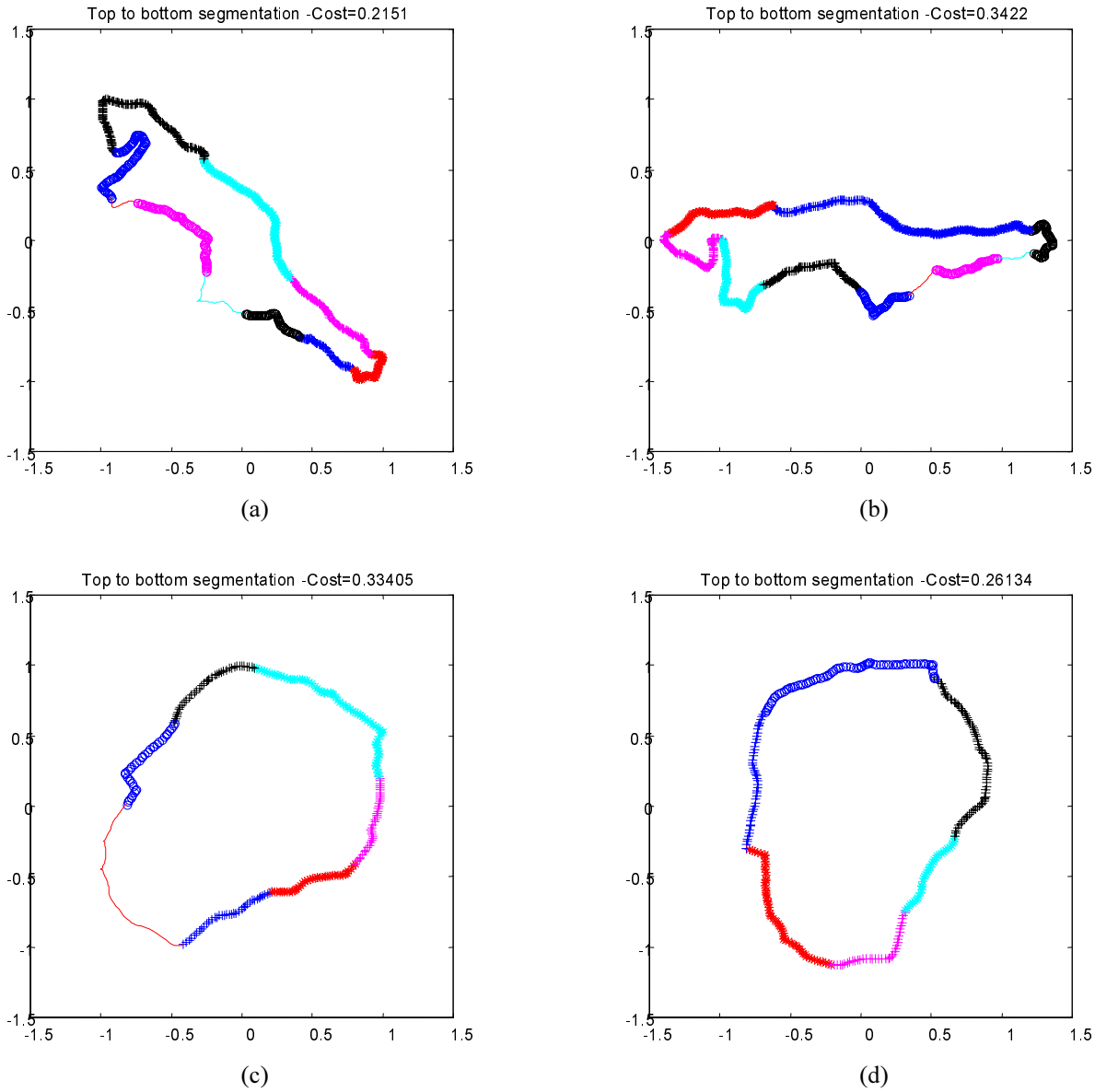


Fig. 4.11: Top-to-bottom segmentation results
(a) – Original boundary, (b) - 45° rotated boundary
(c) – Original boundary, (d) - 36° rotated boundary

As seen from these simulations, the top-to-bottom segmentation algorithm produces different results when segmenting the same object boundary before and after rotation. An optimal segmentation algorithm, relative to any defined cost function, would produce the same segmentation for boundaries before and after rotation. The results shown in Fig. 4.11 indicate that the segmentation result is certainly not optimal. The large differences in the cost function and in the resulting segmentation indicate that the segmentation algorithm is probably far from being optimal.

This page intentionally left blank

4.4.2 Bottom-to-top segmentation

The bottom-to-top segmentation algorithm attempts to merge neighboring segments. Segments are merged as long as the resulting distortion is below the maximal value allowed. The maximal distortion is the only required parameter for this algorithm.

In order to evaluate the results of the bottom-to-top segmentation algorithm, we calculated the value of the cost function C_3 on the resulting segmentation. The values obtained by this function are presented in Fig. 4.12 as the 'cost'.

Simulation results of the bottom-to-top segmentation algorithm are presented in Fig. 4.12. The resulting segmentation before and after rotation is very close, with small differences that arise from the different initialization of the segments.

When comparing the cost of the segmentation obtained for the bottom-to-top segmentation to the cost obtained using the top-to-bottom segmentation, we notice that for the boundaries simulated, the bottom-to-top segmentation has a lower cost. The improved cost is obtained although the bottom-to-top algorithm does not consider the cost function (defined by the top-to-bottom algorithm) in order to make merge decisions.

This page intentionally left blank

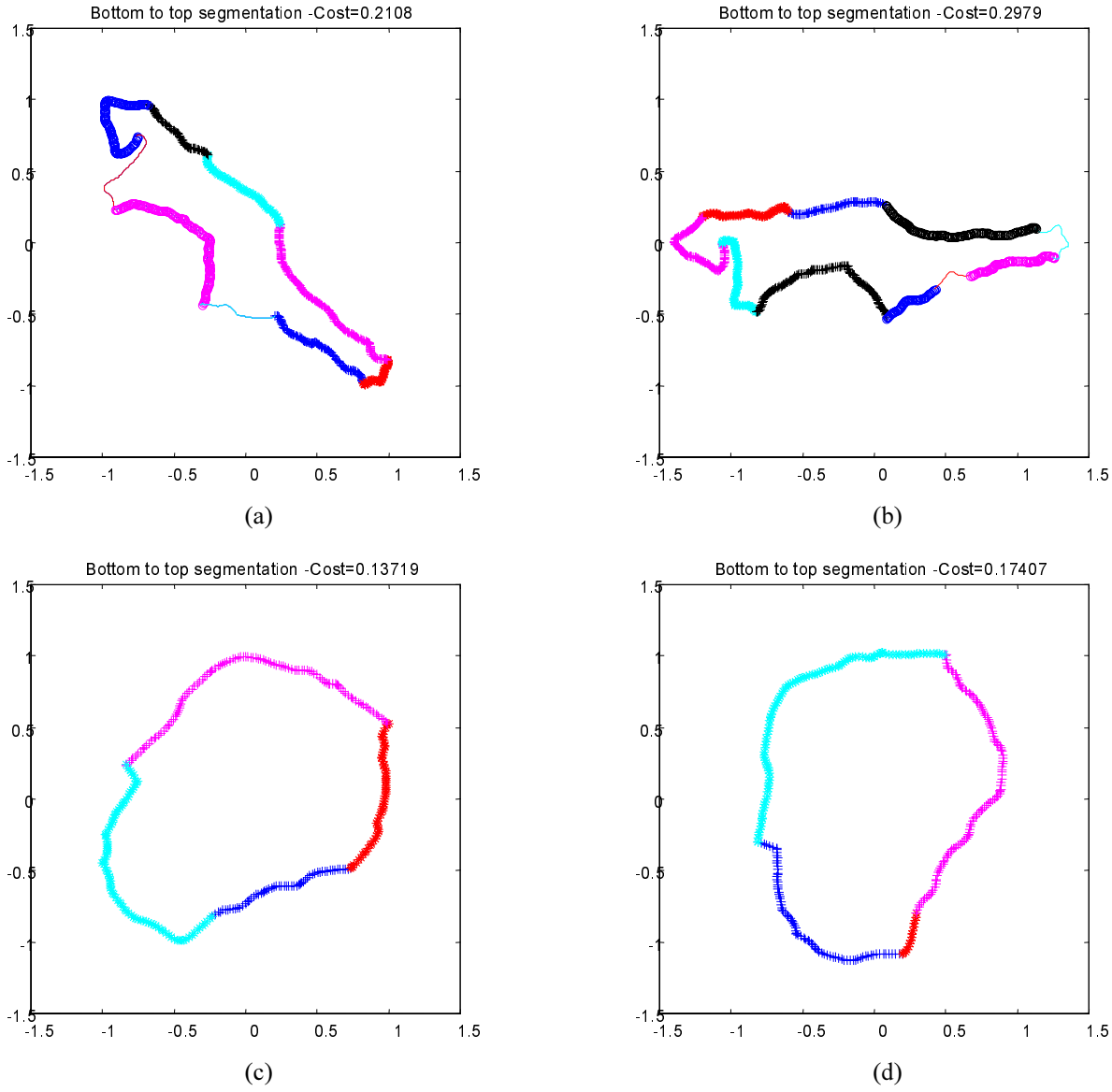


Fig. 4.12: Bottom-to-top segmentation results
(a) – Original boundary, (b) - 45° rotated boundary
(c) – Original boundary, (d) - 36° rotated boundary

From the simulations performed in this section we conclude that the bottom-to-top algorithm, presented in section 4.3.4 is the preferred segmentation algorithm. The top-to-bottom algorithm displayed high sensitivity to rotation, producing highly different segmentations yielding different rate, distortion and cost. The bottom-to-top algorithm produced similar segmentation results (although at different cost values).

Chapter 5 Data Reconstruction

Reconstruction of the input data from the quantized polynomial coefficients is achieved by scanning the zeros of the polynomial where data points existed.

Because there may be polynomial zeros that don't represent data, a filtering mechanism is required. This mechanism includes two parts:

- Forcing a separation between zero-set points that represent original data and spurious zero-set points.
- Scanning the zero-set from a given point (corresponding to an original data point) until reconstruction is complete.

The first part will be incorporated into the fitting algorithm using geometric constraints as described in section 5.1. The second part is accomplished using the continuity of the zero-set, as described in section 5.2. Since the data of a single boundary may be coded using several implicit polynomials, each coding a segment of the contour, a merging scheme is required. This scheme is presented in section 5.3.

5.1 Data reconstruction from IP coefficients

Whereas for object recognition applications, the fitting ends with polynomial coefficients, contour coding requires restorability of the contour data from the coefficients.

In order to restore the data set from the quantized parameters (polynomial coefficients) it should be decided which of the zero-set points represent the original data.

The least-squared (LS) criterion used for fitting the polynomial to the data achieves approximate coverage of the data by the zero-set. This however does not guarantee that all the zeros of the resulting polynomial represent data. The fitting procedure allows for polynomial zeros at locations where there is no data, and there is no error generated by the LS criterion function for these zeros.

This situation prevents us from scanning all of the polynomial's zeros (solving the implicit equation) and considering these zeros as the restored data. Therefore, without changing the algorithm or providing additional information, it is impossible to properly restore the data from the coefficients alone.

5.1.1 Requirements for restoring the data

Fulfillment of the following requirements allows a unique restoration of the data:

- The zero-set of the polynomial has to be continuous along adjacent data points.
- There should be no splits or intersections of the zero-sets of the polynomial.
- A starting point is supplied for restoration.

This set of requirements assures us that if we begin scanning for zeros at the given starting point and move continuously along the zero-set, we can restore the data and stop when we have reached the starting point again.

The fitting algorithms presented in sections 2.2 and 3.4 do not satisfy these requirements and need modifications so that the coefficients could be used to represent the data (as seen in Fig. 5.6 on page 81).

5.1.2 Use of constraints to guarantee restorability

The two first conditions listed above can also be stated as restrictions on the values of the polynomial.

We **require** that the zero-set of the polynomial lie within a thin strip surrounding the data set.

This means that the value of the polynomial around the strip cannot be zero (must be positive or negative for external or internal points respectively).

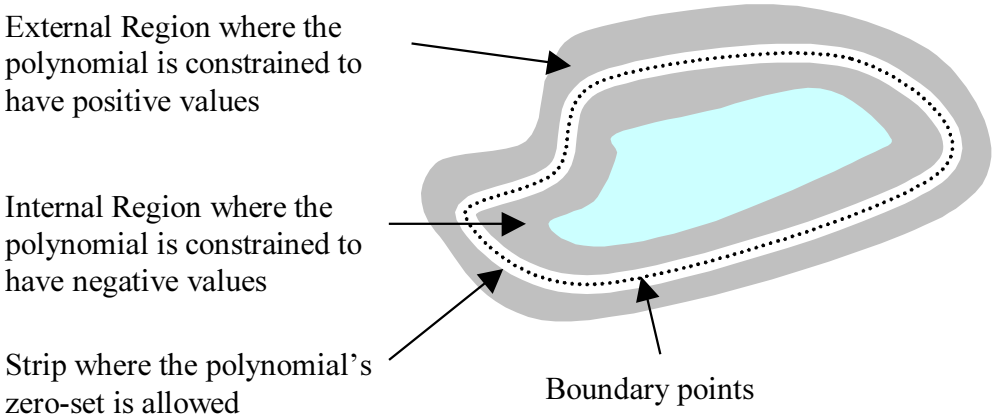


Fig. 5.1: Original data set and constrains

These restrictions, in addition to the fact that the zero-set of the polynomial is a continuous group, can prevent discontinuities and crossings or intersections of the zero-set.

To fulfill these requirements we use the *constrained least squares* solution of the following problem:

- Find $\bar{a}$ which minimizes $E = \bar{e}\,\bar{e}^T$ where $\bar{e} = (\bar{a}M - \bar{b})$, and $M, \bar{b}$ can be generated by any of the fitting algorithms described in section 3.4, subject to:

$$\begin{aligned} &\bar{a}_{LS} M_{EXT} > \bar{0} \\ &\bar{a}_{LS} M_{INT} < \bar{0} \\ &M_{EXT} = \left[\bar{p}^T\left(x_{EXT_1}, y_{EXT_1}\right) \quad \ldots \quad \bar{p}^T\left(x_{EXT_{N-EXT}}, y_{EXT_{N-EXT}}\right)\right] \\ &M_{INT} = \left[\bar{p}^T\left(x_{INT_1}, y_{INT_1}\right) \quad \ldots \quad \bar{p}^T\left(x_{INT_{N-INT}}, y_{INT_{N-INT}}\right)\right] \end{aligned} \tag{5.1}$$

where the constraint points $\left(x_{EXT_1}, y_{EXT_1}\right) \ldots \left(x_{EXT_{N-EXT}}, y_{EXT_{N-EXT}}\right)$ are points external to the strip and $\left(x_{INT_1}, y_{INT_1}\right) \ldots \left(x_{INT_{N-INT}}, y_{INT_{N-INT}}\right)$ are points internal to the strip (see Fig. 5.1).

The result of this set of equations - $\bar{a}_{LS}$ - is no longer a solution of a linear problem (since the space of solutions is limited). We used *Lagrange multipliers* to solve the constrained least squares solution and obtain $\bar{a}_{LS}$ [9].

5.1.3 Using constraints to restrict the maximal error

By using constraints as described above, it is possible to ensure that the maximal fitting error is below a prescribed value.

The distance between the constraint regions that we have constructed and the data is a limit on the maximal error, because the zero-set is bounded by these constraints.

However, we may state constraints that cannot be met and the fitting algorithm would terminate with no valid solution.

If the quality criterion of the fitting procedure is the maximal error, we may improve the fitting by removing contradicting requirements.

When we use constraints that limit the maximal error, we want to obtain the most stable polynomial that meets the constraints (allowed maximal error and restorability).

---

[9] Two Matlab functions can be used for this problem – CONLS, LSQLIN. We used the latter, which provides better results at shorter execution times.

However, the criterion function that we have previously defined (see section 3.4) has two goals:

- Minimal mean squared-error.
- A stable solution.

We see that the MMSE criterion is no longer needed, because we are interested in limiting the maximal error rather than minimize the mean squared error.

Each choice of cost function and constraints produce an optimal solution, which is a point in the subspace (defined by the constraints) that minimizes the cost. When we use only the stability criterion with the constraints, we obtain the most stable solution (in the given subspace defined by the constraints). Therefore, we easily conclude that other criteria that we may use would yield less stable solutions, including our original criterion taking into account both stability and MSE.

After removal of the MMSE criteria ($\sum_{n=1}^{N}\left(\bar{a}\,\bar{p}^{T}(x_n, y_n)\right)^2$), the terms $\bar{b}$ and M in (2.12) are as follows:

$$\begin{aligned} \bar{b} &= \left[\overline{dx} \quad \overline{dy}\right] \\ M &= \left[M_X \quad M_Y\right] \end{aligned} \tag{5.2}$$

where,

$$\begin{aligned} M_X &= \left[\bar{p}_x(x_1, y_1)^T \quad \ldots \quad \bar{p}_x(x_N, y_N)^T\right] \\ M_Y &= \left[\bar{p}_y(x_1, y_1)^T \quad \ldots \quad \bar{p}_y(x_N, y_N)^T\right] \end{aligned} \tag{5.3}$$

5.1.4 Choosing constraint points

In order for the *constrained least squares* solution to be successful a careful choice of the constraint is necessary.

The most important parameter that requires setting is the thickness of the strip that is formed around the data and around which constraint points are being set (see Fig. 5.1). Choosing a wide strip would permit the generation of undesired branches and crossings in the zero set. Choosing a narrow strip would make it impossible for the fitting algorithm to comply with the constraints. We must choose the thickness of the strip so that it is not too wide or too narrow. Of course, every data set requires a different order of the polynomial that can fit the data with the desired constraints. In our simulations the thickness of the strip was determined empirically (see Appendix C for specific details).

Having set the thickness of the strip separating the constraint points, we must decide on a constraint point selection policy. The space in which we wish to limit the polynomial is continuous, and the constraint points that we use for the c*onstrained least squares* algorithm are discrete, so we must sample the continuous space at discrete locations. The sampling policy needs to be decided by the user, so that the goal of the constraints can be fulfilled with minimal computational complexity.

The examples shown in section 3.5, were produced using the following sampling scheme (as shown in Fig. 5.2):

- Create two regions in the plane. One region where internal constraints are confined to, and the other where the external constraints are confined.
- Inside these two regions, sample the constraint points at random distances with an average distance of 5 pixels[10].

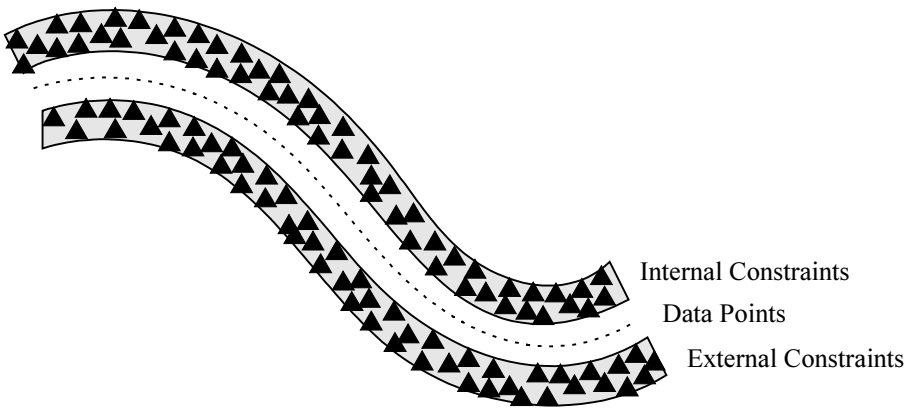


Fig. 5.2: Selection of constraint points

The result of this constraint points sampling scheme are two sets of points, internal and external to the data set points, with a density that disables crossing by zero sets. The density of the constraint points was determined experimentally.

[10] This sampling scheme was derived empirically. The distance between sampling points was set so that no reconstruction errors occurred in our simulations.

5.2 Zero-set scan

The mechanism of constraint fitting, implemented during the fitting process (as described in the pervious section), forces the zero set of the polynomial to be without intersections along the data. Therefore, when we supply a point on the zero-set that represents a data point, the zero-set may be scanned and data points are reconstructed.

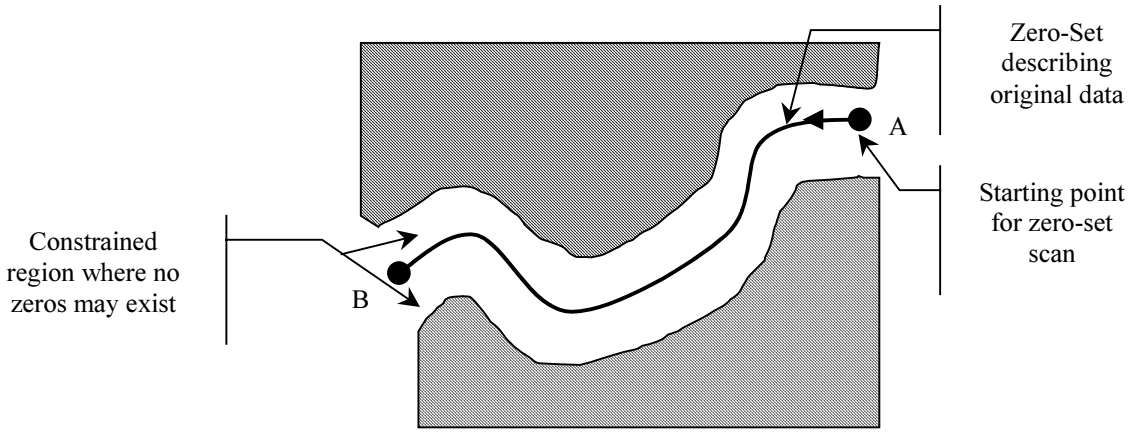


Fig. 5.3: Zero-set scan of constrained polynomial

To find the zero-set points of the polynomial from point A (on the zero-set) to point B, the following can be done:

- Initialize the first data-set point as A: $\bar{z}(n) = A$
- Move from the current position in a direction tangent to the polynomial's zero-set[11] at a distance 'd':

$$\bar{z}_T(n+1) = \bar{z}(n) + d \cdot \bar{T}(n) \tag{5.4}$$

where $\bar{T}(n)$ is a vector tangent to the polynomial's zero-set at point $\bar{z}(n)$:

$$\bar{T}(n) = \frac{\left(\frac{\partial P_{\bar{a}}(\bar{z}(n))}{\partial y}, -\frac{\partial P_{\bar{a}}(\bar{z}(n))}{\partial x} \right)}{\sqrt{\left(\frac{\partial P_{\bar{a}}(\bar{z}(n))}{\partial x} \right)^2 + \left(\frac{\partial P_{\bar{a}}(\bar{z}(n))}{\partial y} \right)^2}} \tag{5.5}$$

The derivatives of the polynomial are calculated according to (3.19) and (3.20) on page 24.

[11] The direction perpendicular to the tangent - (x, y) is $(y, -x)$.

- Find the closest point on the zero set to $\bar{z}_T(n+1)$:

$$\bar{z}(n+1) = SD_{\bar{a}}(\bar{z}_T(n+1)) \tag{5.6}$$

where the operator $SD_{\bar{a}}(p_0)$ represent a solution for the zero-set point closest to point p_0 [12].

- Repeat until the zero-set point $\bar{z}(n)$ reaches the required end point B.

5.3 Segment merging

In this section we present an algorithm for merging the zero-sets of the IPs representing neighboring segments into a single curve – describing the original, unsegmented curve. The operation of this algorithm relies on the constraints that were used for the fitting of the IP.

In order to merge neighboring segments, making up an object's boundary, we must find the points at which each two neighboring segments are connected. To accomplish this, each segment's reconstruction begins at its center and continues outwards the edges until an intersection with a neighboring section is encountered. The constraints imposed during the fitting process (see section 5.1) assure us that scanning along the zero-set would produce true reconstructed data with no false intersection with spurious zero-set branches.

Fig. 5.4 demonstrates a boundary and its reconstruction from four segments.

[12] We used the *steepest-descent* algorithm to find the closest zero-set points, as described in Appendix D.

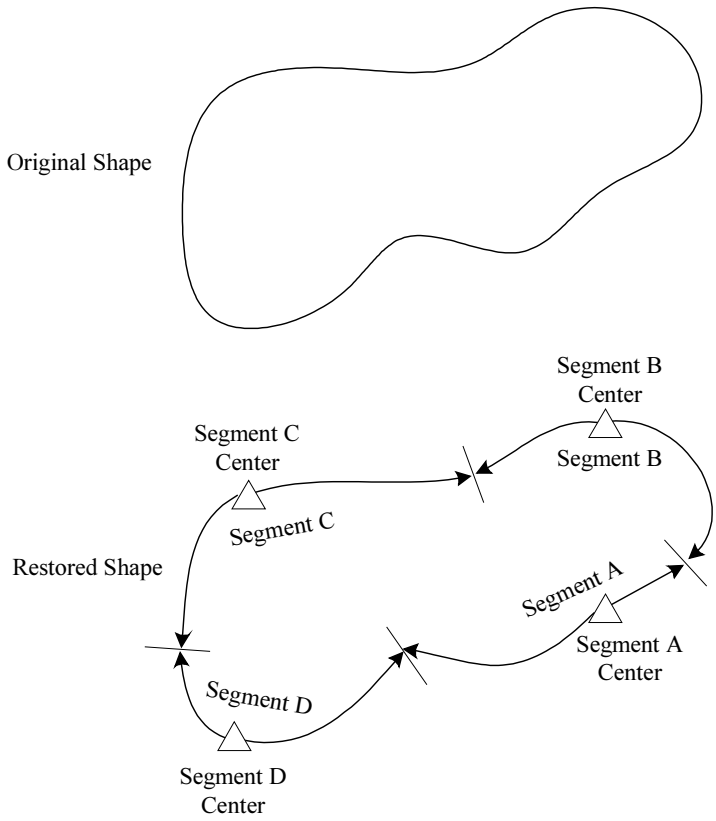


Fig. 5.4: Object boundary and its reconstruction from four segments

The reconstruction of all segments making up an object's boundary is performed simultaneously. The base point for the reconstruction of each segment is its center, which is part of the side information for each segment. The zero-set points are scanned in both sides of the center.

The two sides of each segment are denoted here as "positive" and "negative" branches. These branches are scanned in increments of $d \cdot \overline{T}(n)$ and $(-d) \cdot \overline{T}(n)$ respectively, where $\bar{z}_p(n_p)$ and $\bar{z}_n(n_n)$, with $\overline{T}(n)$ as described in (5.5), are the zero-set points on the positive and negative branches respectively.

At first, both sides of each segment are marked is "incomplete". At each iteration, all incomplete segments are "grown", i.e. a point is added to each branch according to its scan direction.

After each iteration, when all the incomplete segments are grown, all segment branches are checked for intersection with neighboring branches. Since the segments are grown with resolution of 'd', an intersection is declared when points from two segments are at a distance of 'd' or less. Positive growing branches are checked against the negative growing branches of the neighboring segment and vice versa. When an intersection if found, the intersecting branches of both segments are marked as complete. Also, since it is not guaranteed that the intersecting branches meet "head to head" (currently scanned points of both branches), all points from the intersection point to the end of each branch are deleted.

This process continues until all segments are marked "complete", as seen in Fig. 5.5.

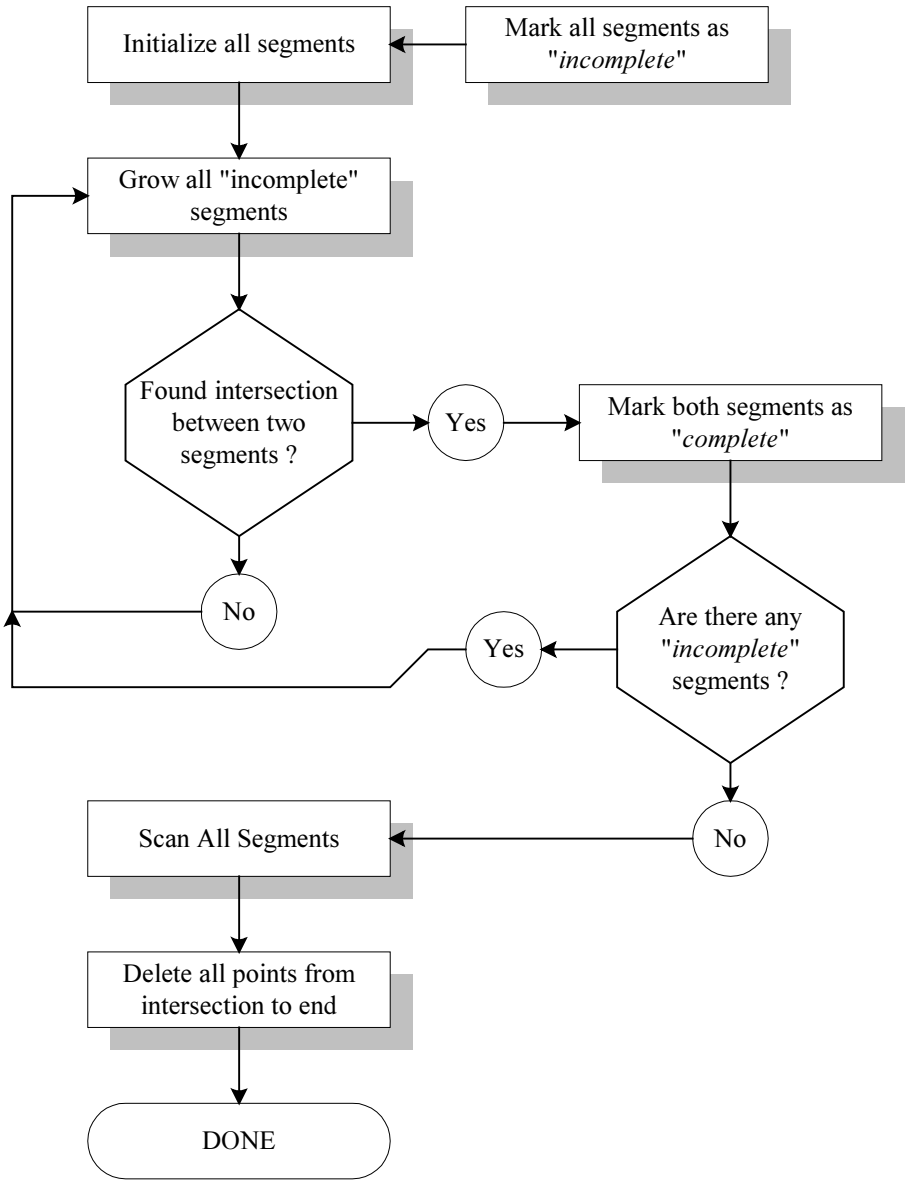


Fig. 5.5: Flow chart of curve segments merging

5.4 Simulation results

Here we present the result of polynomial fitting with no quantization. The purpose of this simulation is to demonstrate the effectiveness of the constraints presented in section 5.1 and it's importance when using the data reconstruction algorithm presented in section 5.2.

The fitting is done using the different algorithms presented in Chapter 4 and extended in this chapter using constraints. Each algorithm returns a parameter vector that best fits its criterion. This vector does not exactly satisfy the criterion and some errors remain (non-compliance between the desired result and the optimal feasible result). The size of these errors depends on the coefficients' errors and the sensitivity to these errors.

In the next two sections we examine the effects of different criterion functions on the fitting results.

The fitting was performed using the original *3L* algorithm, *Min-Max* and *Min-Max constrained* algorithms.

Examples of fitting 14^{th} and 8^{th} order polynomials are given in Fig. 5.6 and Fig. 5.7, respectively. An example of data reconstruction from polynomial coefficients is given in Fig. 5.8.

Fitting 14th Order polynomials

Two object boundaries are shown (top and bottom rows in Fig. 5.6).

It is possible to see the significant improvement in the fitting results going from the left column (*3L*) through the center column (*Min-Max*) to the right column (*Min-Max constrained*).

Polynomials of high order (here 14^{th}) exhibit high sensitivity to noise. The following examples clearly demonstrate the influence of the sensitivity function on the quality of the solution.

The improvement in the fitting results is achieved by fitting with the improved criterion (moving from the left column – Fig. 5.6(a1), Fig. 5.6(b1) to the center column – Fig. 5.6(a2), Fig. 5.6(b2)).

This page intentionally left blank

When implementing the constraints we can see that the data has no intersections or crossing and therefore can be easily recovered (going from the center column – Fig. 5.6(a2), Fig. 5.6(b2) to the right column – Fig. 5.6(a3), Fig. 5.6(b3)).

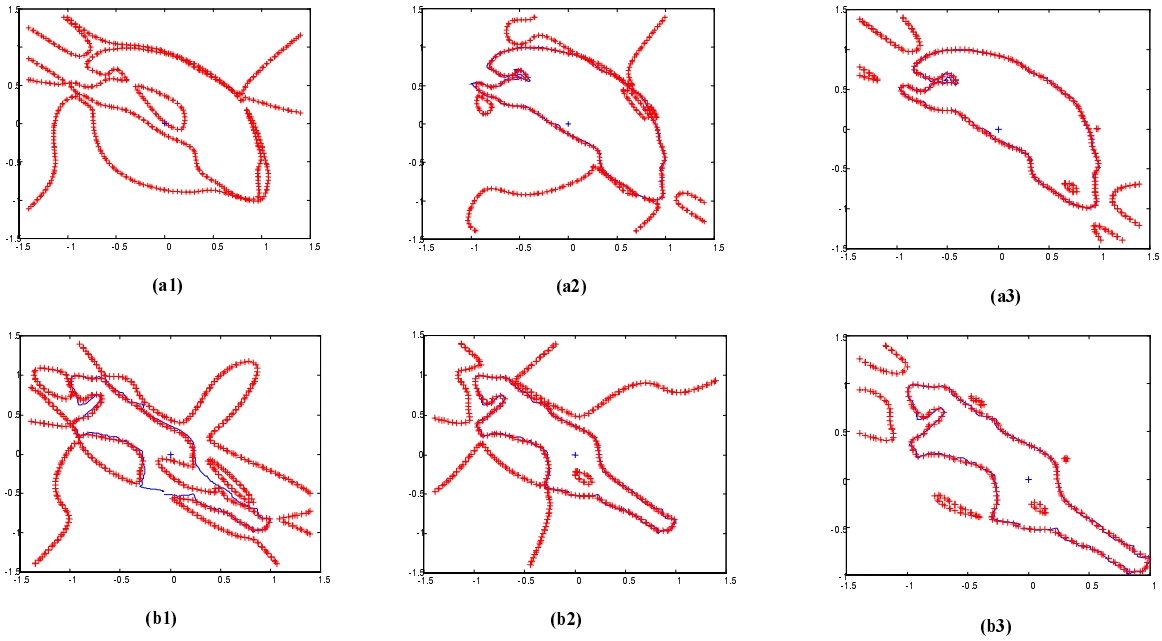


Fig. 5.6: Fitting results using *3L*, *Min-Max* and *Min-Max constrained* algorithms
Top – Data set of boundary 1, Bottom – Data set of boundary 2
Left – *3L* algorithm, Center – *Min-Max* algorithm, Right – *Min-Max constrained* algorithm
Solid blue line displays original data.
Red crosses display zero-set of the polynomial.

This page intentionally left blank

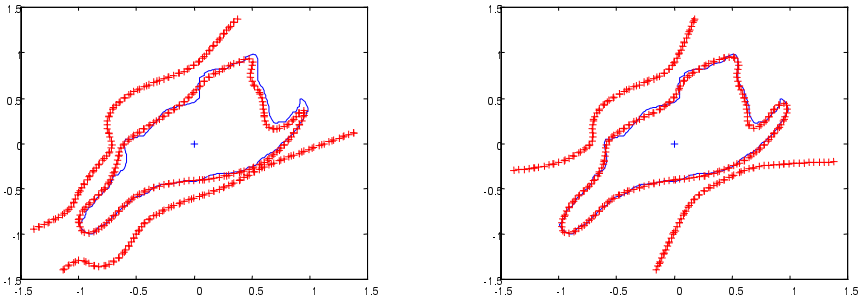
Fitting 8th Order polynomials

An object boundary is shown in Fig. 5.7.

A smaller change in the fitting quality (relative to order 14 polynomials presented above) can be seen.

Since polynomials of smaller degree are inherently less sensitive to errors in the coefficients (relative to 14^{th} order polynomials), the fitting results obtained by using the *Min-Max* algorithm have only improved a little relative to the *3L* algorithm.

No constrained solution is presented here because there are no crosses and intersections in the unconstrained solution in Fig. 5.7(a2). In this case the results of both the unconstrained and the constrained solutions overlap.



(a1) **(a2)**

Fig. 5.7: Fitting results using *3L* and *Min Max* algorithms
Left – *3L* algorithm, Right – *Min-Max* algorithm
Solid blue line displays original data.
Red crosses display zero-set of the polynomial.

This page intentionally left blank

Reconstruction of data from the coefficients of 14th order polynomials

Using the constrained least squares solution we obtain a polynomial whose zero set can be used for representing object boundaries for image compression applications. Fig. 5.8 compares the resulting polynomial fit and consequent restored data of the same boundary using both unconstrained and constrained fits.

As seen from these simulations, although the *Min-Max* algorithm produces a polynomial with better fitting to the data relative to the *3L* algorithm, the data is properly restored only by using the constraints of the *Constrained Min-Max algorithm*.

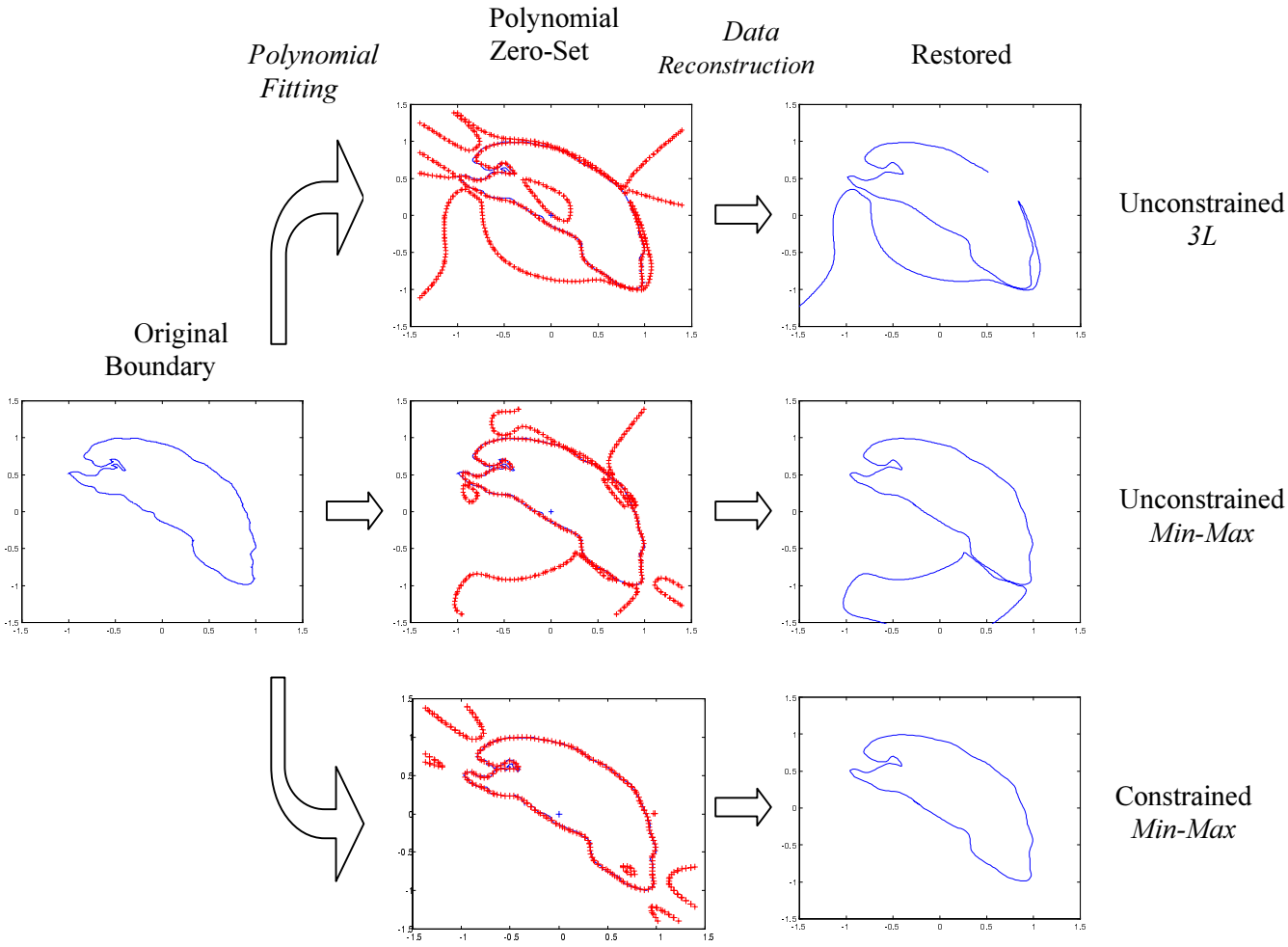


Fig. 5.8: Boundary reconstruction results
Left image – Original boundary
Center column – 14th order fitted polynomial
Right column – Restored data using zero set tracking

This page intentionally left blank

Chapter 6 Contour coding

The previous chapters described polynomial fitting, curve segmentation and data restoration. This chapter describes the combination of these topics into a complete contour-coding scheme.

The complete contour-coding scheme, based on IPs, is summarized in Fig. 6.1.

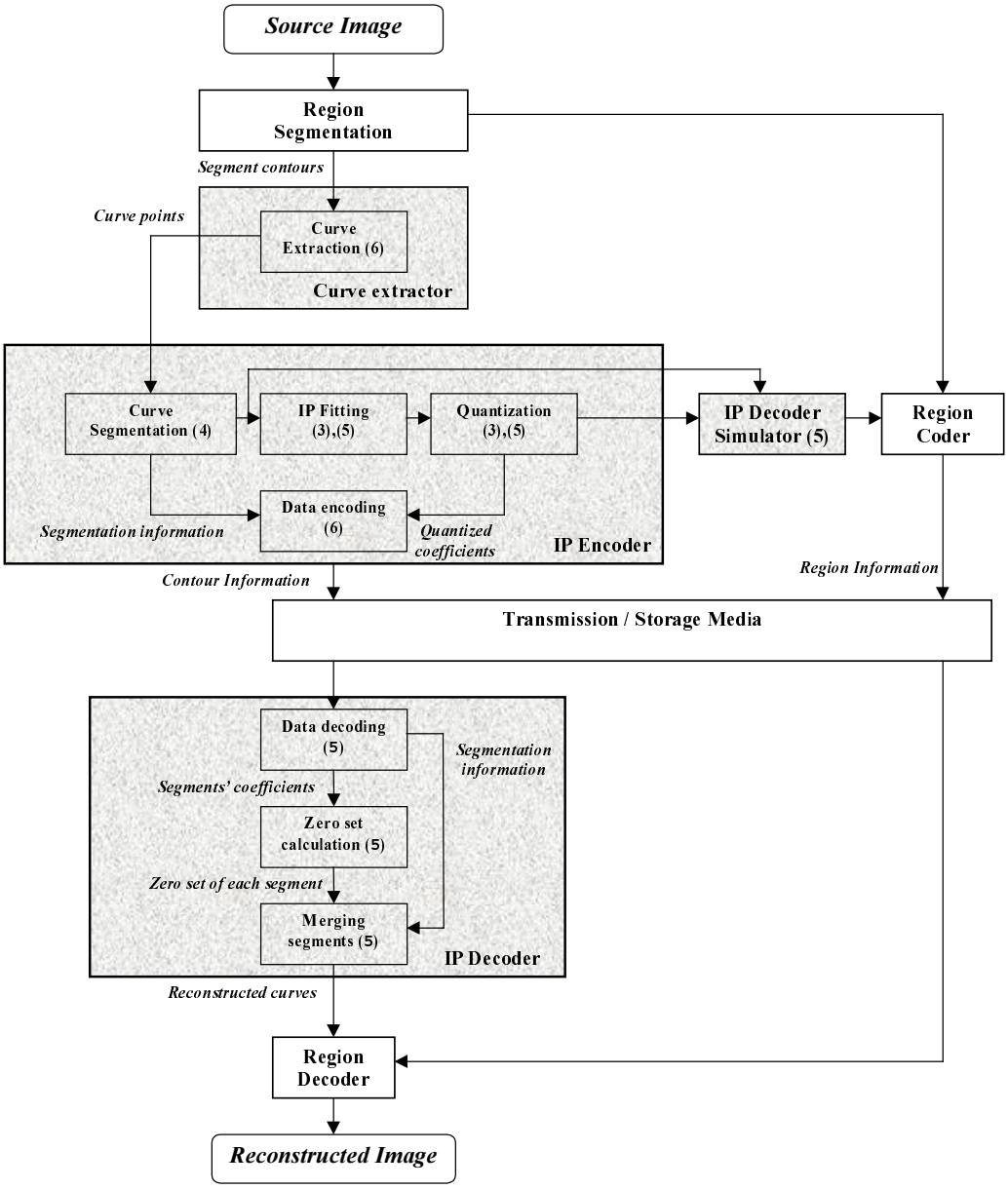


Fig. 6.1: Block diagram of contour coding scheme based on IPs.
Chapter numbers where each block is described appear in parentheses

The boundary information of a segmented image[13] is extracted (see section 6.1). Each boundary curve is further segmented into more efficiently coded curve segment (see Chapter 4). Than, an implicit polynomial is fitted to each curve segment (see Chapter 3). This fitting is performed according to the geometric constraints described in section 5.1 so that the boundary information can later be restored.

The coefficients defining each IP are quantized. Prior to quantization, all the coefficients of each IP are scaled so that the largest absolute value of the coefficients is 1. This scaling allows the allocation of a fixed number of MSB bits to each coefficient with no shifting. Of course, this scaling does not change the zero-set of the polynomial. The quantized coefficient information, along with some necessary side information such as polynomial order, starting point for zero-set scan and data normalization information (see section 4.1) is packed together and makes up the code.

At the decoder, the zero-set of the polynomials is used to reconstruct the coded data. A scan is performed on the zero-set (see section 5.2), the segmented curves are merged and the data is reconstructed.

6.1 Fitting objects with shared boundaries

Each two adjacent objects in an image share boundaries. For a coding application, coding the shared boundaries twice – once in each object – leads to redundant information and to a higher code rate. On the other hand, segmenting object boundaries at the points where neighbors change (see Fig. 6.2) may lead to inefficient segmentation which, in turn, may also lead to an increased code rate. The purpose of this section is to compare the two options for coding object boundaries: as complete boundaries, or as segments shared by two neighbors.

[13] The segmentation of the source image is to regions. The input of this coding scheme is the contour information of the segmented image.

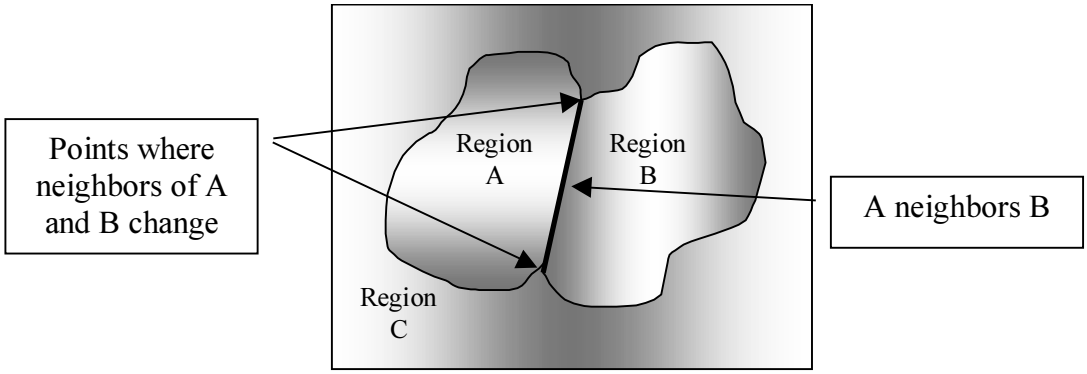


Fig. 6.2: Three regions in an image

6.1.1 Coding complete object boundaries

According to this approach the segmentation of each object boundary begins with a single segment containing all the points on the object's boundary (see Fig. 6.3).

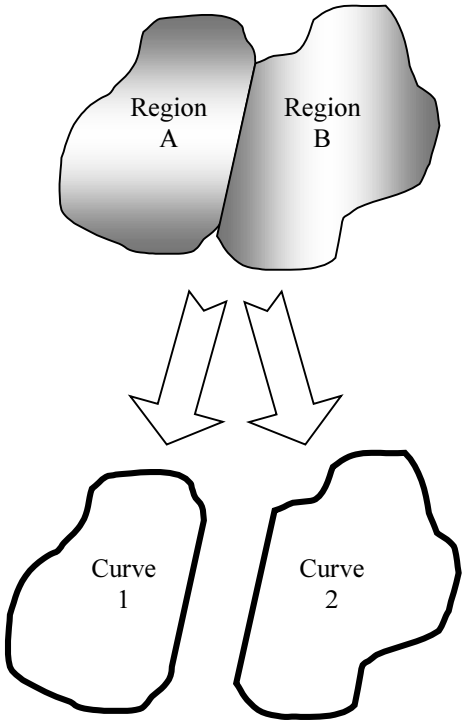


Fig. 6.3: Region and boundary curves of two objects (A,B)

Segmentation of the boundaries of the image in Fig. 6.3 would begin with the two boundaries of A and B. Each boundary would be further segmented in order to reduce the rate / distortion cost.

When coding each object by itself, conflicts may arise on the borders between neighboring objects. This is because the fitting process of implicit polynomials yields a representation that has errors relative to the original data. Thus the shared part of the

boundary of (A) and that of (B) may differ, although they both represent the same border. The reconstruction algorithm should be aware of these errors (gaps or overlaps) and should be able to resolve the conflicts that are caused by them. Although the reconstructed image may have errors relative to the original image, there must not be any gaps or overlaps between objects.

6.1.2 Coding shared curves

It is possible to take advantage of the redundancy that exists in object boundaries (two objects sharing borders). We can segment the boundaries into curves, so that each curve belongs to two objects (or to the outer border). An example is shown in Fig. 6.4.

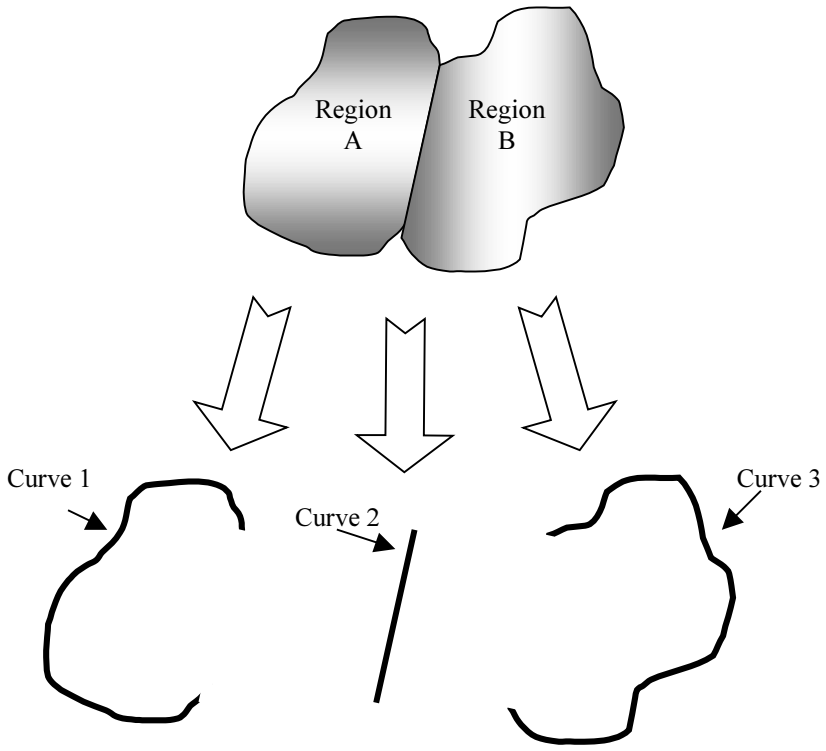


Fig. 6.4: Complete boundary and shared curves of two objects (A,B)

Segmentation of the boundaries of the image in Fig. 6.4 would begin with three boundaries – the border between A and B (shared curve by A and B) and the two curves each containing the boundaries or A and B excluding the shared curve.

Since the curves that are shared by two objects are coded only once, the possible conflicts between two boundaries are eliminated. This method can also lead to a more efficient representation since redundant data is not coded. However, small objects neighboring large objects may cause a segmentation that is not optimal for the large

objects. This results from the fact that change of neighbors may lead to curve segmentation points, which may not be chosen by the curve segmentation algorithm (see section 6.1.3 for a comparison between the two methods).

When coding shared curves, the process in Fig. 6.5 is performed.

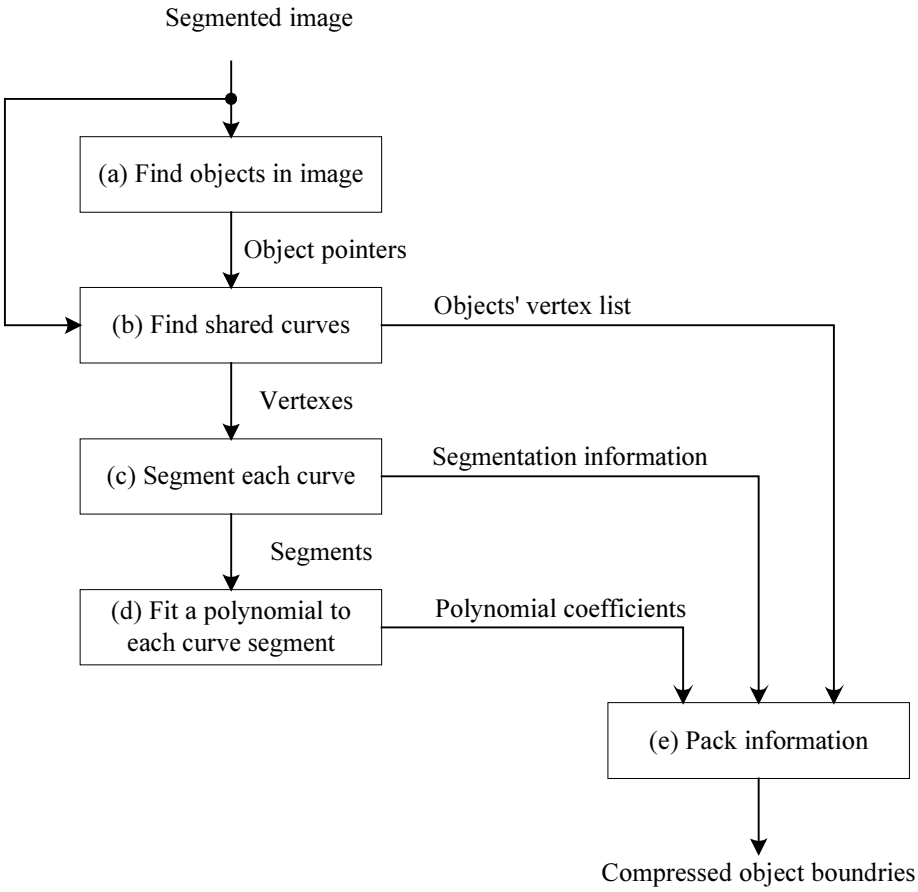


Fig. 6.5: Flow chart for curve coding using implicit polynomials

First, objects must be recognized in the segmented input image (block (a)). Second, the objects in the image are separated into curves (block (b)). The curve information needs to be coded for the storage / transmission. This information is necessary for recovery of the image by the decoder. Each curve is further segmented in order to achieve a more efficient representation by implicit polynomials (block (c)). The segmentation information is also required at the decoder in order to merge the curve segments into complete curves. Finally, each segment of each curve is fitted by an implicit polynomial (block (d)). The coefficients of these polynomials and the side information required for restoring the data (see section 4.2) are packed together (block (e)) and sent to the decoder.

For example, object A in Fig. 6.4 would be represented as [curve 1, curve 2] and object B would be represented as [curve 2, curve 3]. All the coded information – polynomial coefficients, curve segmentation information and object's curve list should be considered when calculating the rate. Of course, the data for the representation of curves 1-3 is transmitted only once and used for the appropriate object contours.

6.1.3 Comparison of the two methods

Apparently, boundary representation using shared-curves is preferable over complete boundaries curve extraction. This seems to be the case since the later scheme produces twice the curve data relative to the first. However, the data redundancy does not necessarily indicate a less efficient representation, and since it turns out that there are images for which one approach is better, and other images for which the other approach is better.

For example, consider the image in Fig. 6.6:

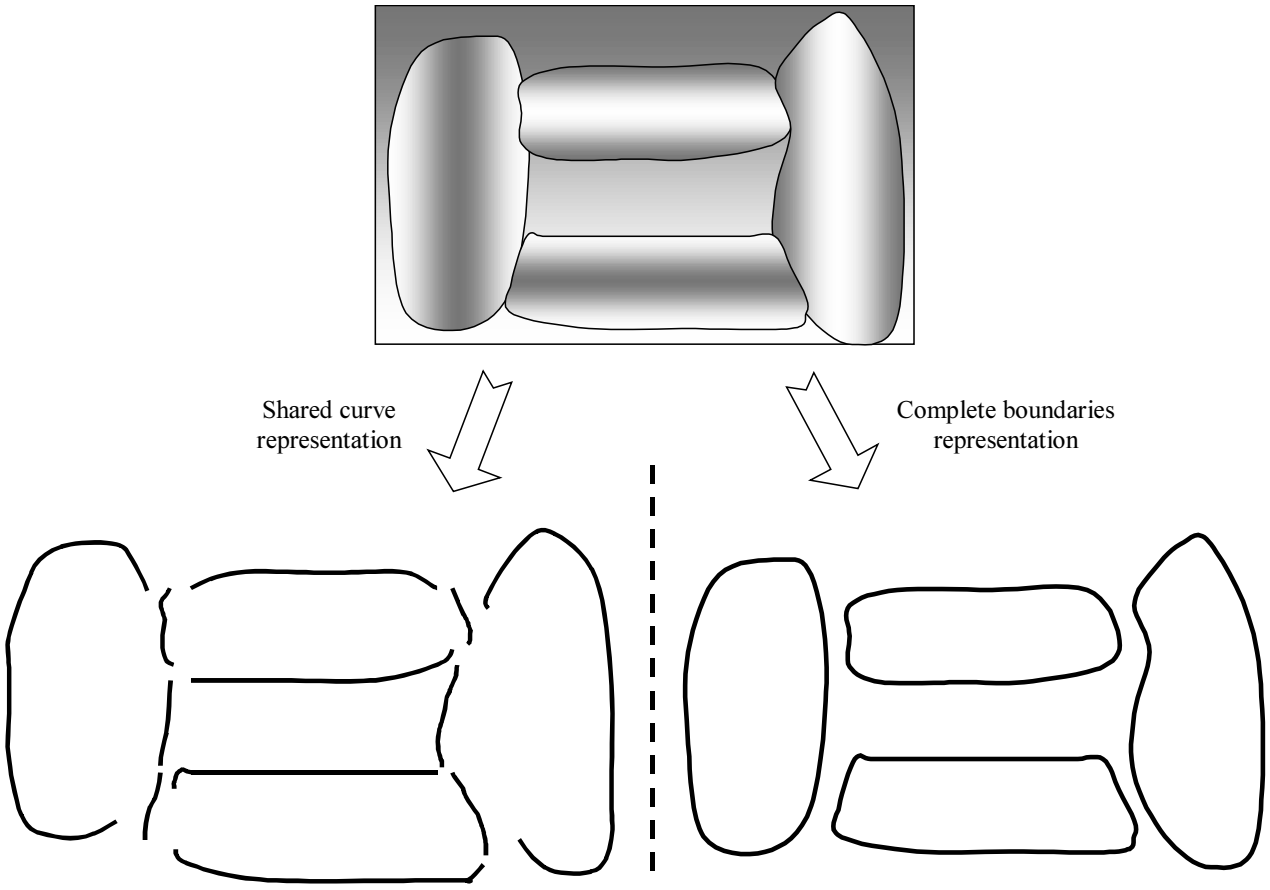


Fig. 6.6: Curve extraction sample 1
Image whose representation is more efficient using complete boundaries curve extraction

The example in Fig. 6.6 shows an image that breaks into ether 12 curves shared by neighboring objects, or into only 4 curves using complete boundaries curve extraction. Since the complete boundaries in this example are smooth it should be possible to represent them efficiently using a low order polynomials. However, breaking these bodies into 12 small curves would make it necessary to use more polynomials, with more side information, and only a small profit in the complexity (order) of the polynomials.

This situation is typical when the image contains large regions that can be efficiently coded using low order polynomials that represent large contours. Arbitrarily breaking these contours into smaller curves (not based on rate-distortion considerations) would not contribute much to reduce the order of the required polynomials, but would make it necessary to code more polynomials and more side information.

Images that contain objects that are difficult to describe using a single polynomial are better coded using shared curves. The image in Fig. 6.7 shows an example where this approach produces better coding results.

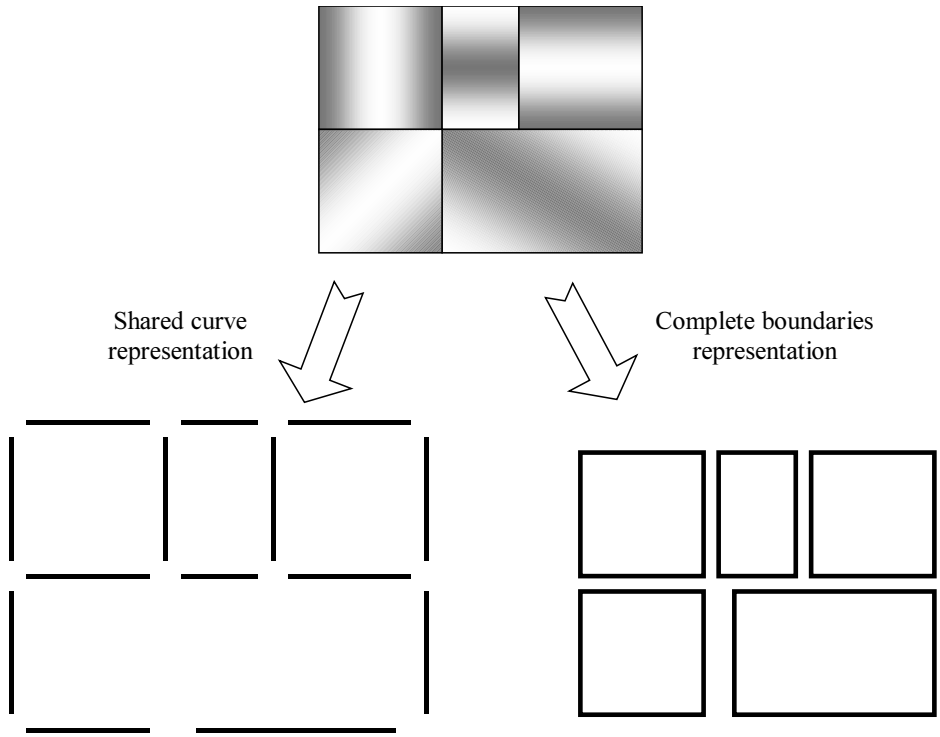


Fig. 6.7: Curve extraction sample 2
Image whose representation is more efficient using shared curves

The image in Fig. 6.7 is more efficient to code using shared curves. The reason for this is because the complete rectangles would be further segmented later, so that 1st order polynomials could describe each segment. The segmented rectangles would generate redundant information where two curves would describe the same boundary between neighboring regions.

In summary, we can state that some images may be better coded using one curve extraction method while other may be better coded using the other method. An adaptive algorithm can employ either extraction methods depending on the objects found in the image.

6.2 Hybrid contour coding

Throughout this work we used implicit polynomials to represent boundaries. IPs can efficiently describe smooth boundaries with a large number of data points. Rugged boundaries require a high order polynomial, and when rugged boundaries contain a small number of points it is inefficient to describe them using IPs.

In order to use the IPs efficiently and obtain good compression results, we suggest to separate the input curves into curves that are well coded and curves that aren't well coded using IPs. This is a hybrid coding scheme where the curve dichotomy results determine the amount of curve data represented by IPs and the overall compression ratio of that data. This process is described by the block diagram in Fig. 6.8.

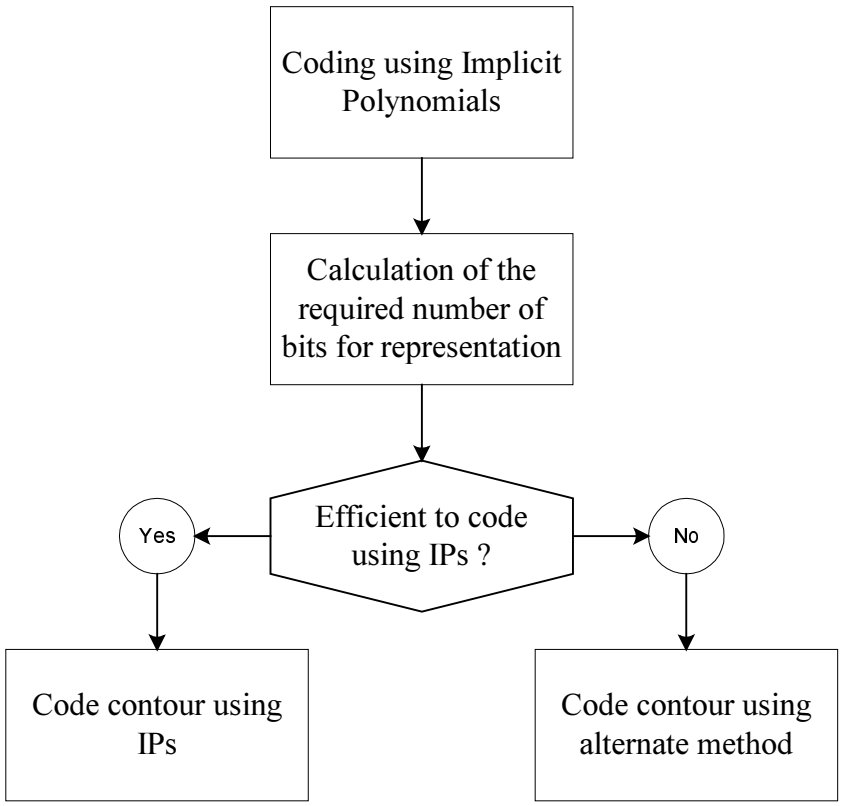


Fig. 6.8: Classification of contours according to preferred coding method

6.3 Simulation results

In this section we present simulation results of coding complete images. The purpose of these simulations is to demonstrate the complete contour compression and reconstruction algorithms. For this simulation we a used computer animation image, a segmented medical image (Fig. 6.15) and the well known peppers image (Fig. 6.16).

The image in Fig. 6.9 was obtained as a B/W image from a collection of animation objects. Its graphical simplicity makes it useful for recognition of image quality after reconstruction. Several simulation runs were performed using this image, yielding different compression ratios for different permissible distortion values, as seen in Fig. 6.10. As a reference for comparison, the same image was compressed using both four and eight neighbors chain-codes, requiring 10818 and 13358 bits respectively.

The reconstruction results obtained at different distortion values are presented in Fig. 6.11, Fig. 6.12, Fig. 6.13 and Fig. 6.14.

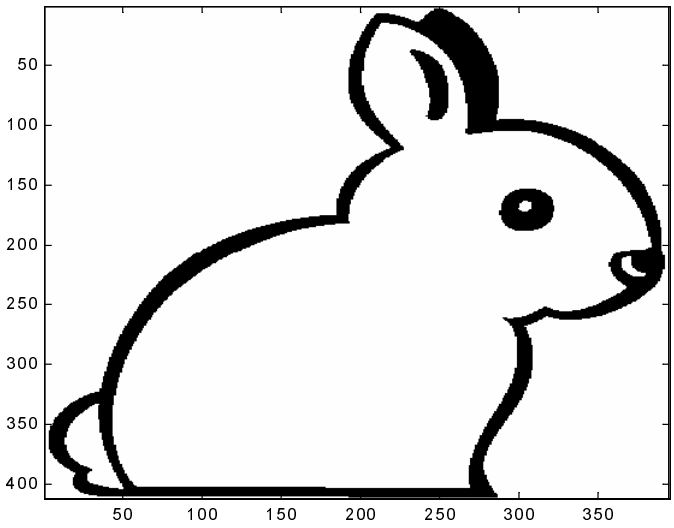


Fig. 6.9: Original *Rabbit* image
X and Y axes indicate pixels

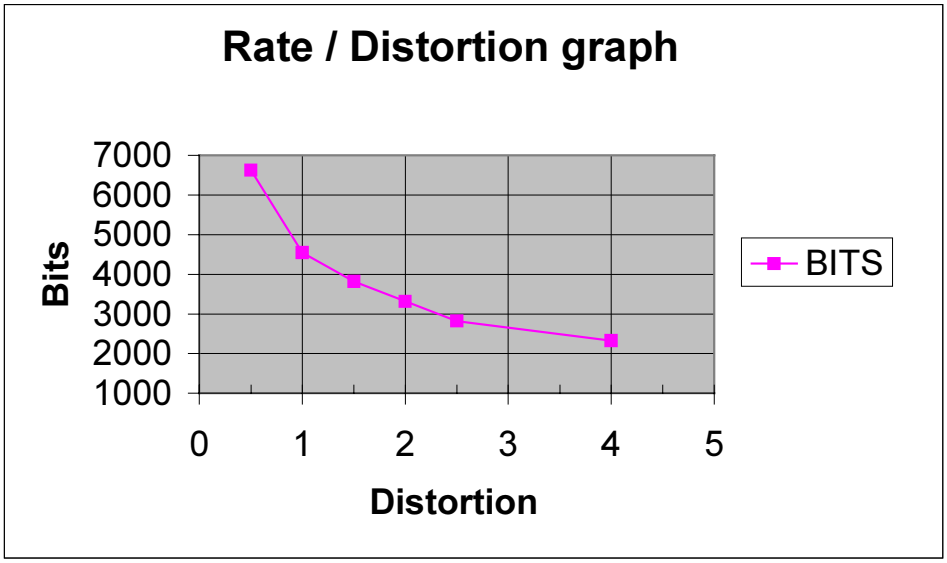


Fig. 6.10: Rate vs. Distortion graph of *Rabbit* image
Rate – bits required for representation
Distortion – maximal error in pixels

This page intentionally left blank

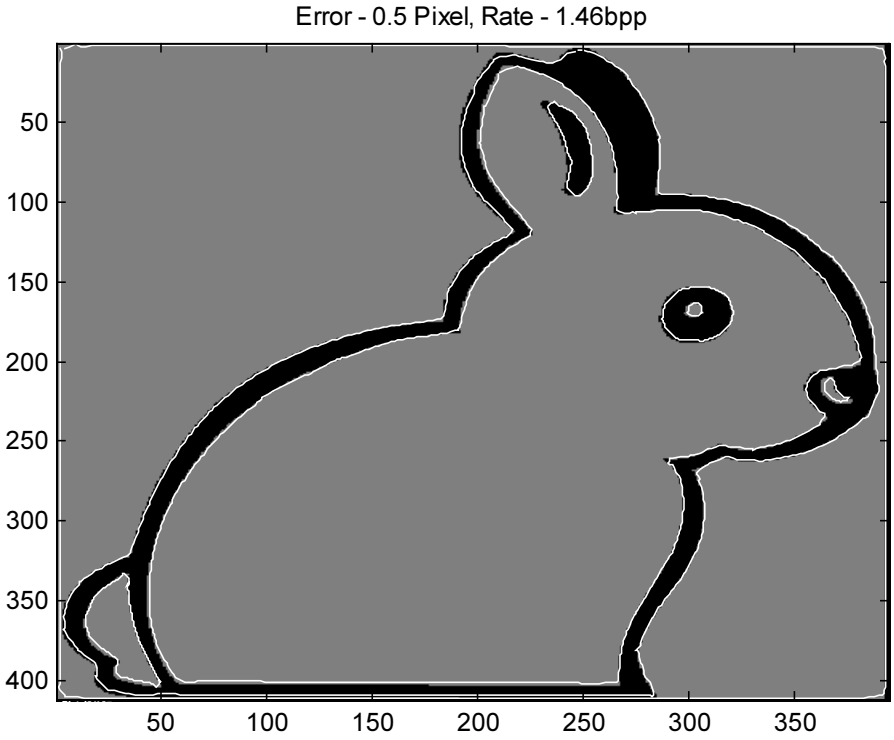


Fig. 6.11: Reconstruction result of encoded *Rabbit* image – Distortion: 0.5 pixel, Rate: 1.46bpp
4,446 contour points extracted using 8 neighbors chain-code

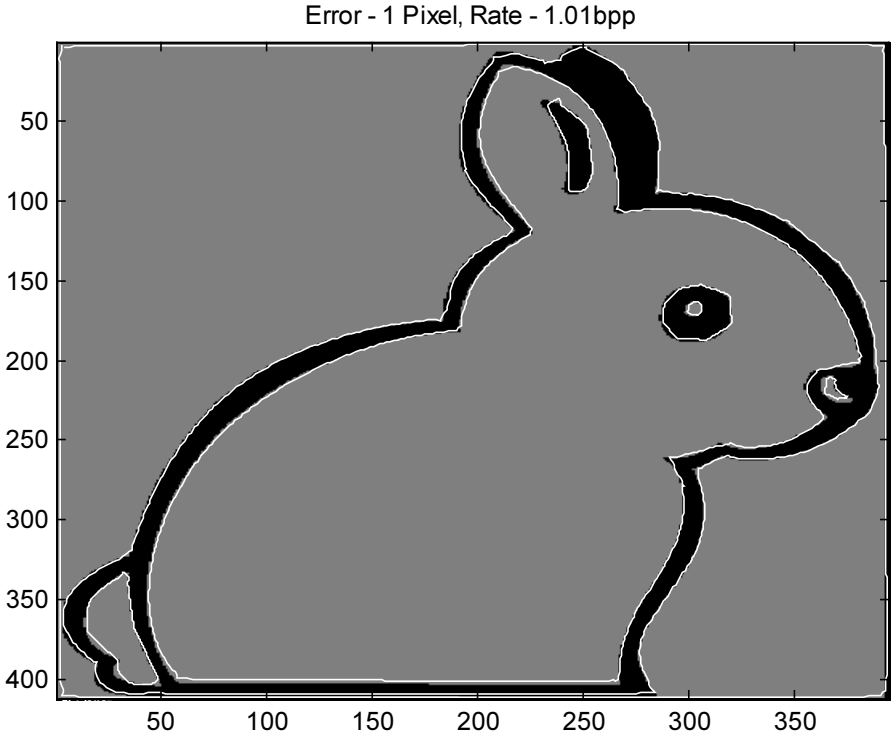


Fig. 6.12: Reconstruction result of encoded *Rabbit* image – Distortion: 1 pixel, Rate: 1.01bpp
4,446 contour points extracted using 8 neighbors chain-code

This page intentionally left blank

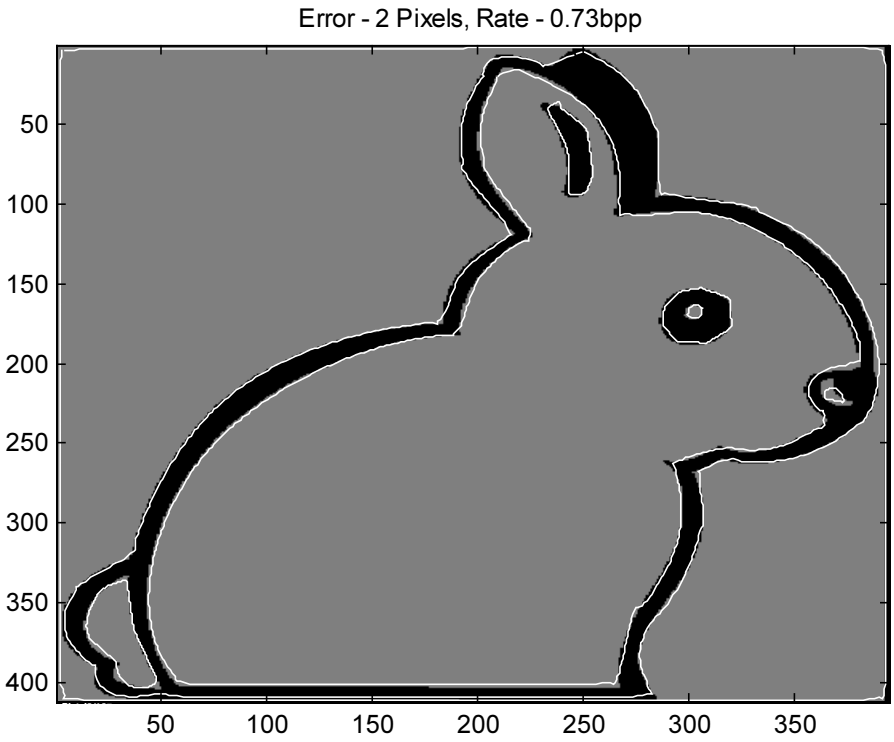


Fig. 6.13: Reconstruction result of encoded *Rabbit* image – Distortion: 2 pixels, Rate: 0.73bpp
4,446 contour points extracted using 8 neighbors chain-code

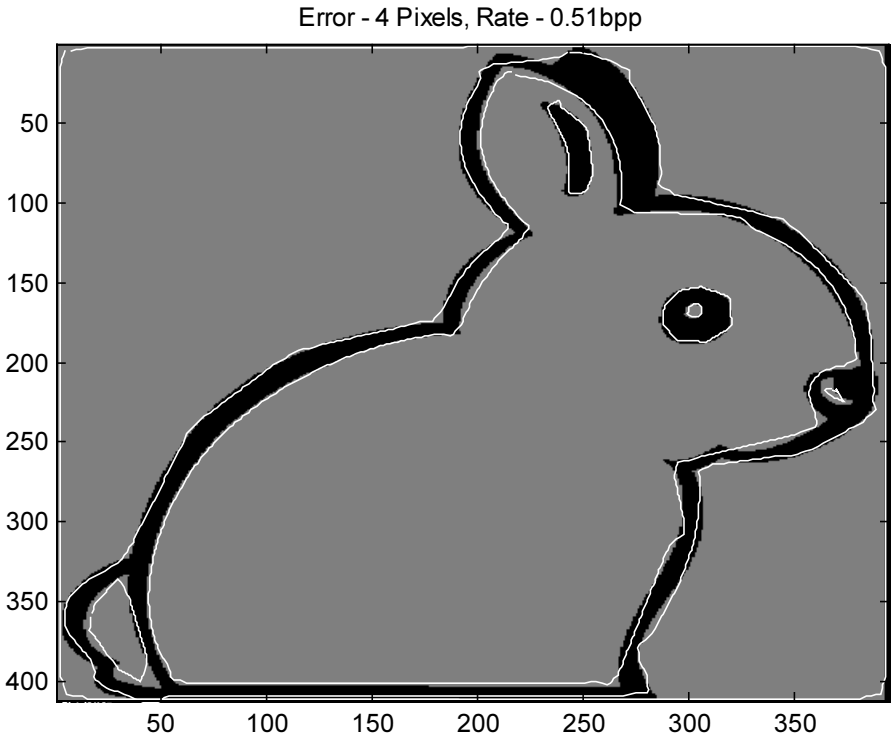


Fig. 6.14: Reconstruction result of encoded *Rabbit* image – Distortion: 4 pixels, Rate: 0.51bpp
4,446 contour points extracted using 8 neighbors chain-code

This page intentionally left blank

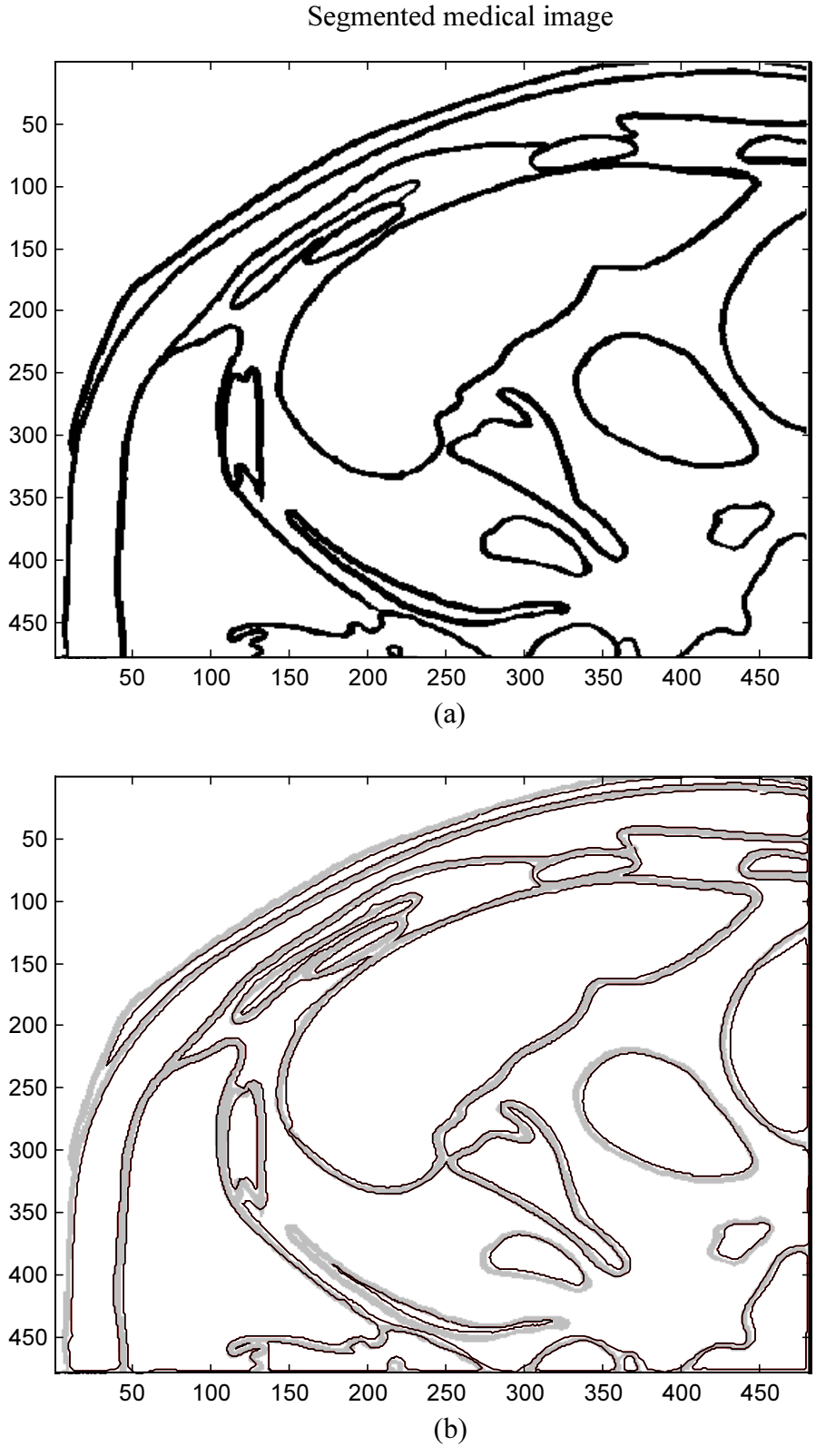


Fig. 6.15: Reconstructed contours of medical image
(a) – Original image (after segmentation), (b) – Reconstructed image after IP compression
In (b) – thin black lines represent IP reconstructed data, thick gray lines represent original contours
Number of pixels: 11,378, Maximal distortion, 1.5 pixels, Rate: 0.89 bpp

This page intentionally left blank

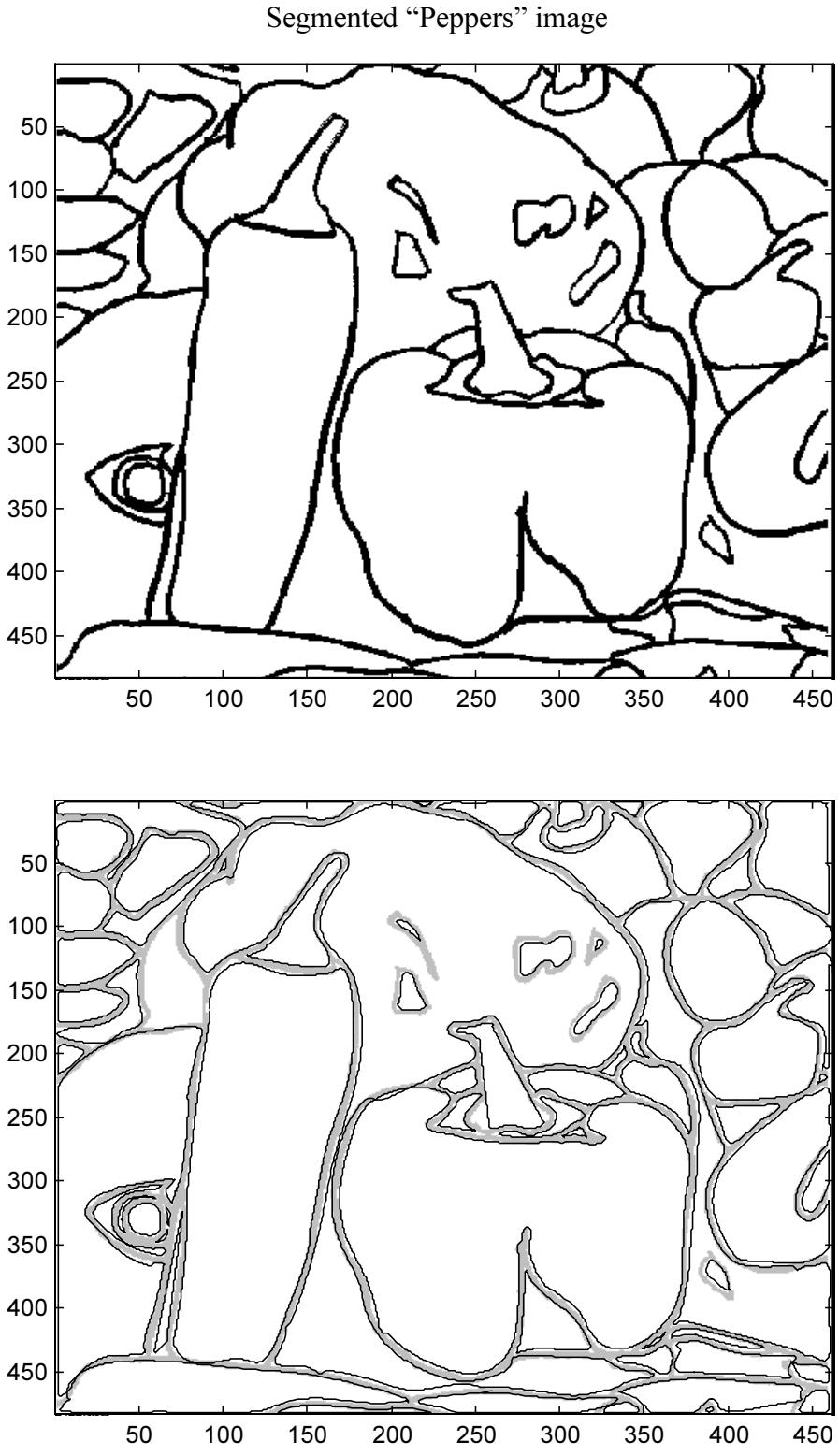


Fig. 6.16: Reconstructed contours of “peppers” image
(a) – Original image (after segmentation), (b) – Reconstructed image after IP compression
In (b) – thin black lines represent IP reconstructed data, thick gray lines represent original contours
Number of pixels: 13,868, Maximal distortion : 1.5 pixels, Rate: 1.21

This page intentionally left blank

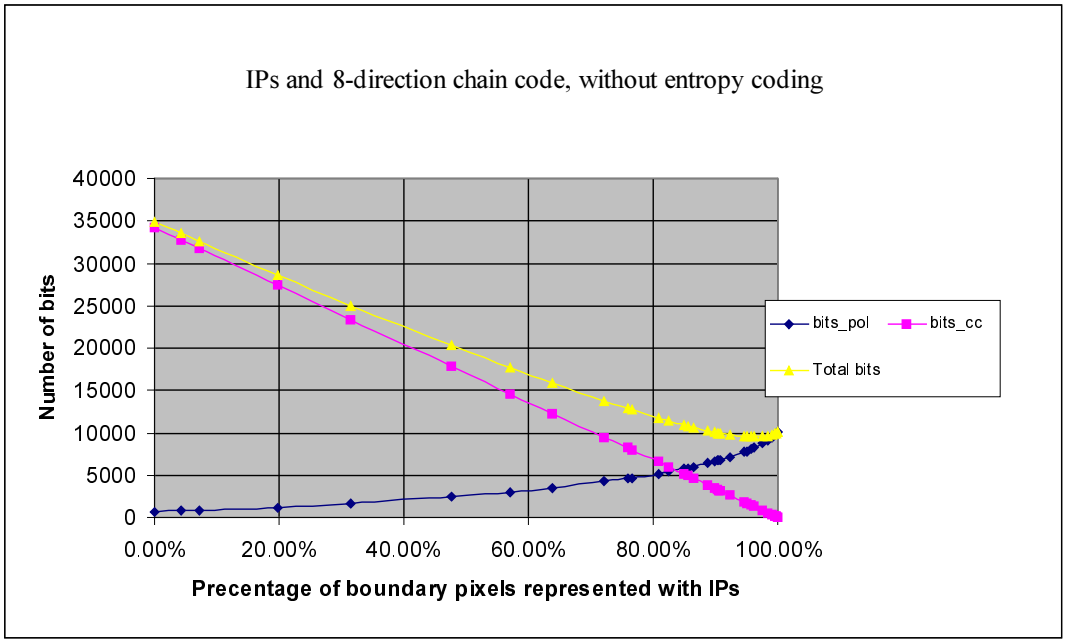
The different points on the graphs in Fig. 6.17 and Fig. 6.18 correspond to different dichotomies of the input curves. When using IPs to represent curves with very high representation efficiency only, the algorithm produces very high compression ratios for the data points represented. Including more curves that are more difficult to represent (require more bits) using IPs, the average rate per pixel rises.

The optimal point to choose on the graph depends on the bit rate required by the alternate coding method. For example, eight-neighbors chain coding with no further compression, requires 3 bits per data point. Additional entropy coding of the resulting chain code may bring the bit rate to 1.5 bits, depending on the curve shape.

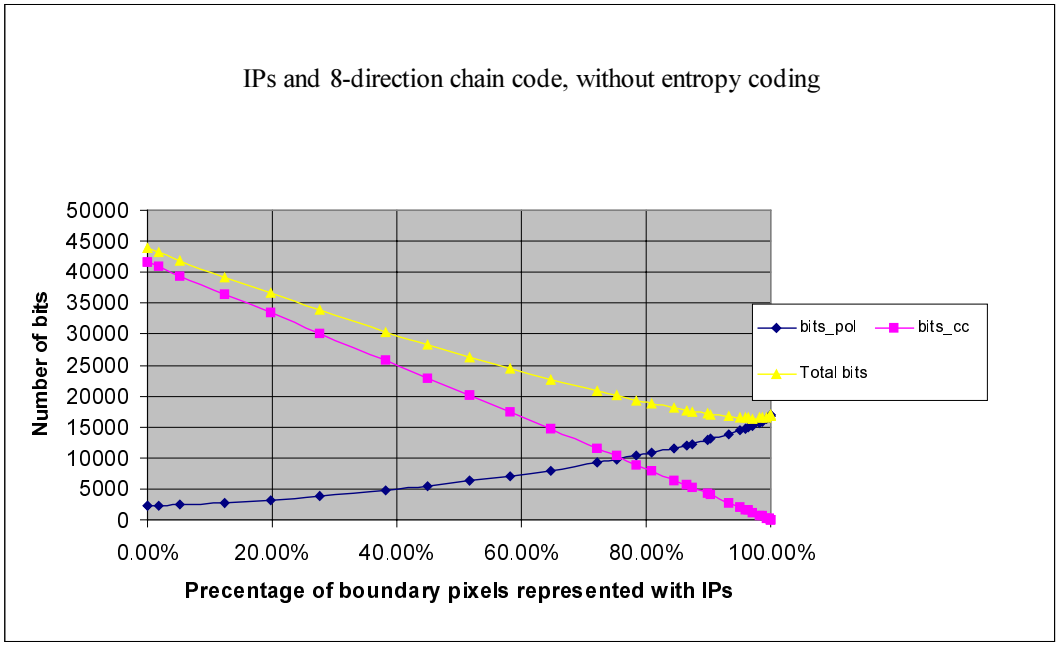
Fig. 6.17 and Fig. 6.18 show the required bit rate for hybrid coding using chain code without entropy coding and with entropy coding, respectively. As seen from these graphs, the most efficient representation is obtained when coding about 90% of the data using IPs and the rest of the curve information using chain code. Other contours with different details may have a different optimal dichotomy.

Using the hybrid coding scheme presented in section 6.2, the graphs in Fig. 6.17 and Fig. 6.18 are obtained.

This page intentionally left blank



(a) Required no. of bits for medical image. Distortion: 1.5 pixels, No. of pixels: 11,378

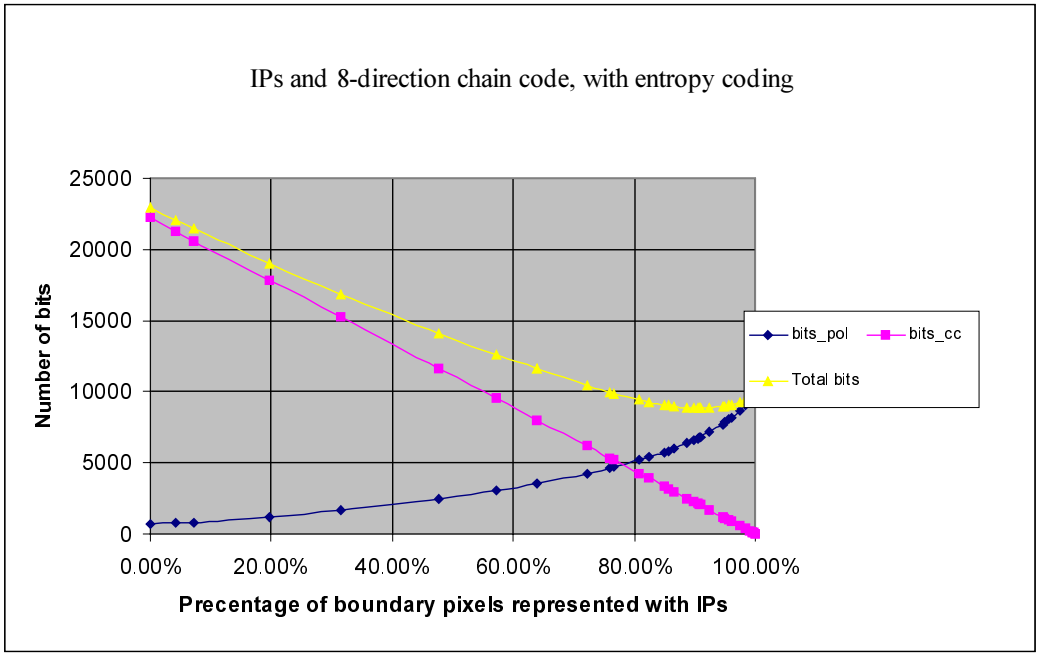


(b) - Required no. of bits for "peppers" image. Distortion: 1.5 pixels, No. of pixels: 13,868

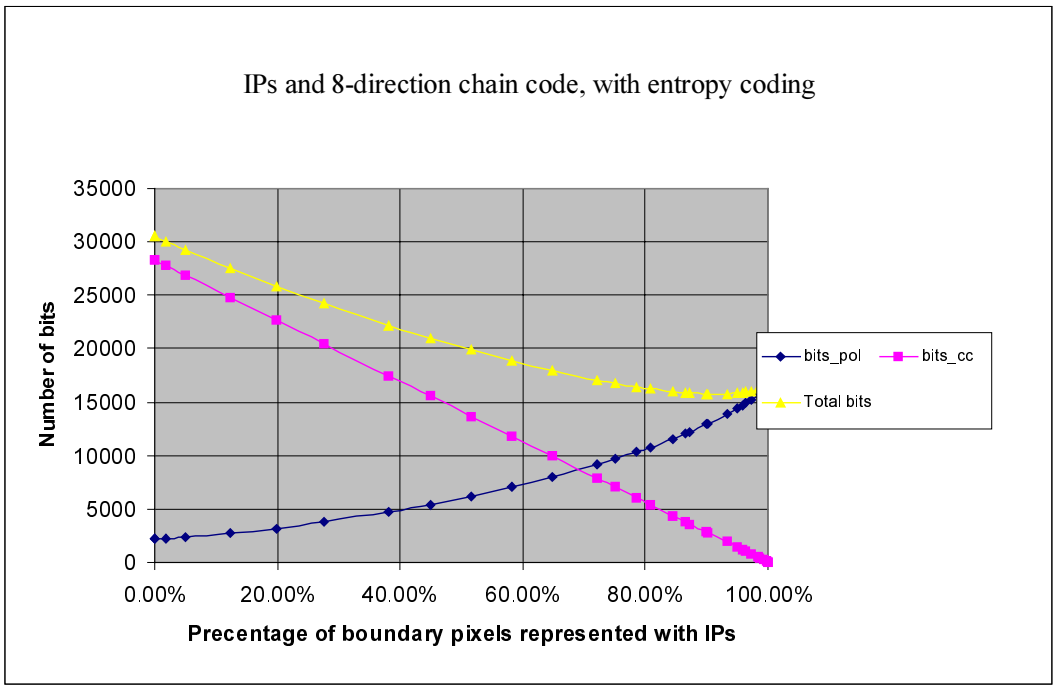
Fig. 6.17: Rate versus percentage of data represented by IPs without entropy coding
IPs fitted with maximal error of 1.5 pixels, chain code **without entropy coding**
Rate in pixels: Blue line - IP rate; Purple line - chain-code rate; Yellow line - overall rate
(a) – medical image with 11,378 pixels
(b) –"peppers" image with 13,868 pixels

This page intentionally left blank

This page intentionally left blank



(a) Required no. of bits for medical image. Distortion: 1.5 pixels, No. of pixels: 11,378



(b) - Required no. of bits for "peppers" image. Distortion: 1.5 pixels, No. of pixels: 13,868

Fig. 6.18: Rate versus percentage of data represented by IPs with entropy coding
IPs fitted with maximal error of 1.5 pixels, chain code **with entropy coding**
Rate in pixels: Blue line - IP rate; Purple line - chain-code rate; Yellow line - overall rate
(a) – medical image with 11,378 pixels
(b) –"peppers" image with 13,868 pixels

This page intentionally left blank

Chapter 7 Summary and conclusions

This work presents a robust fitting algorithm of IPs to object contours and proposes its usage for contour coding in the context of image processing applications. Since implicit polynomials were previously applied to recognition applications, this work had to deal with both theoretical and practical aspects of boundary coding via implicit polynomials.

Chapter 3 provided most of the theoretical basis for the work related to the stability of implicit polynomial zero-sets. This chapter also presented several simulation results demonstrating the properties of implicit polynomials fitted to object contours using the proposed algorithm. It was demonstrated that the resulting robustness of implicit polynomials greatly depends on the criterion used for fitting, and that the fitting algorithm should be adopted for the needs of the application.

Chapters 4 and 5 address practical aspects of the coding scheme. These chapters present more heuristic solutions for practical problems and are tested using simulations.

Curve segmentation handled in Chapter 4 stands out as a complex problem. Our understanding, due to the special properties of implicit polynomial zero-sets, is that all segmentation decisions should be based on the fitting quality of implicit polynomials to the curve segments. The solutions proposed in this chapter are only a small subset of possible solutions to the segmentation problem. Having concluded that an optimal exhaustive search is impractical (and practically impossible) for even short curves, we presented two approaches to solve the problem. Naturally, the segmentation algorithm is the most sensitive part of the work to different cost functions. Whereas the polynomial fitting algorithm is generically designed to provide the most robust polynomial, the segmentation is driven by required rate and distortion properties. Higher distortion allowed would produce larger segments that would be loosely fitted with a large fitting error and would require only a small number of bits for representation. A tighter distortion requirement would drive a finer segmentation and the usage of higher order polynomials, resulting in smaller fitting errors at the expense of a higher bit count.

Chapter 5 provides an algorithmic solution for the reconstruction of data fitted by constrained implicit polynomials. The results of this chapter, addressing the restorability of coded curves from polynomial coefficients, complement the development of the fitting algorithm described in Chapter 3.

Chapter 6 describes the integration of the previous chapters (3,4,5) into a complete boundary coding system and provides simulation results of complete images coded using implicit polynomials. Results presented in this chapter show the potential of IPs in coding image contours when very low bit rates are needed while most of the bits are consumed by the contours (like in animation). Of course, the high computational complexity associated with representation of object boundaries using IPs, limit their use to such special applications where efficient coding of boundary information is critical.

To summarize, we conclude that the fitting algorithm presented in this work can generate implicit polynomials that are useful for a good representation of object boundaries. This representation can be useful for a variety of applications, among which are object recognition and, with the addition of constraints presented here, also for contour data compression.

7.1 Future work

There are several aspects that could be further studied and would contribute to this subject area.

The previously mentioned curve segmentation problem presented in Chapter 4 could be further explored. We experimented with several cost functions for the top-to-bottom curve segmentation, and a single distortion measurement for both the top-to-bottom and bottom-to-top segmentation algorithms. Other options may be attempted for both.

The constrained-fitting solution briefly mentioned in section 5.1, could be improved using other, more efficient, optimization methods. Also, it is possible that some of the parameters that were obtained empirically could be analytically found using error bounds obtained after fitting IPs with the *Min-Max* algorithm. These error bounds, as described in section 3.2.2, may be used to set the constraint-region parameters described in section 5.1.4.

We used a simple quantization scheme for the coding of the polynomial coefficients and side information produced by the algorithm. According to this scheme the coefficients of the polynomials are rounded to the nearest integer. More efficient information coding techniques may include further compression of the coefficients and of the side information using entropy coding [25].

Further improvements in the performance of the implicit polynomial-based encoder are expected if the properties of the polynomials would be taken into account in the design of the region segmentation of the coded image. The input for the algorithms presented here is the result of the segmentation performed using the morphological algorithm presented in [16]. This algorithm is oriented towards the efficient coding of region information, while chain coding is used to represent contour information. It is possible that a segmentation that is based on implicit polynomials for region segmentation (implicit polynomials used to define contours) would yield segments that are very efficiently coded using implicit polynomials. We demonstrated that image contours can be efficiently coded using implicit polynomials, even after the image was segmented with no consideration given to the contour coding scheme presented here. Therefore, it is reasonable to assume that such segmentation algorithms providing both efficient region coding and contour coding using implicit polynomials are achievable.

Future work using implicit polynomials for contour compression could also be applied to image sequences, rather than still images. As done in current video compression standards, the redundant information common to consecutive image frames in a sequence is not coded, and only differences between images need to be coded. Applying 3D implicit polynomials, rather than the 2D polynomials used in this work, could assist in coding smoothly changing information in image sequences. Considering the time (or frame counter) as the 3^{rd} dimension, 3D polynomials can be fitted to objects found in sequences. Objects whose contours are smoothly varying during a sequence would be well described by a 3D polynomial.

For example, when an object's distance to the camera changes, a single 3D polynomial can capture and describe the shape of the object in the entire sequence. However, using chain code to describe the object's contour, the object contour in each frame would have to be coded separately.

A 2D 4^{th} degree polynomial has 15 coefficients, and a 3D 4^{th} degree polynomial has 35 coefficients. If the sensitivity of the 3D polynomial remains the same as the

sensitivity of the 2D polynomial, then both would required the same number of bits per coefficient. In that case, coding an object found in three consecutive frames would be beneficial by using a 3D polynomial (since $3 \cdot 15 > 35$). In cases where an object is found in several dozens of frames it may be very attractive to code its contour using a single 3D polynomial, even at the cost of an increased bit count per coefficient (in order to maintain the same quantization error).

Appendix A – Derivation of Taubin's fitting algorithm

This appendix follows the steps of the development of Taubin's original fitting algorithm. There are two versions of this algorithm. The original method is described here, along with some formulations and approximations that are also useful for the understanding of error properties of implicit polynomials.

The final version of the fitting algorithm, used by Taubin, is described in section 2.2.1.

We denote the closest point on the zero-set of the implicit polynomial described by $\bar{a}$, to the data point (x_n, y_n) as:

$$\left(z_{\bar{a},x\_n}, z_{\bar{a},y\_n}\right) = \underset{(x,y)\in Z_{\bar{a}}}{\arg\min} \left(\sqrt{(x-x_n)^2 + (y-y_n)^2} \right) \tag{A.1}$$

Since the direction of the line connecting $\left(z_{\bar{a},x\_n}, z_{\bar{a},y\_n}\right)$ and (x_n, y_n) is parallel to the direction of the gradient of the polynomial at $\left(z_{\bar{a},x\_n}, z_{\bar{a},y\_n}\right)$, an expression for this distance can be written according to the following[14], where $\langle \bar{a}, \bar{b} \rangle$ denotes an inner product between the vectors $\bar{a}$ and $\bar{b}$:

$$\begin{aligned} dist\left(Z_{\bar{a}}, (x_n, y_n)\right) &= \left\langle \left[\left(x_n - z_{\bar{a},x\_n}\right) \quad \left(y_n - z_{\bar{a},y\_n}\right) \right], \frac{\nabla P_{\bar{a}}\left(z_{\bar{a},x\_n}, z_{\bar{a},y\_n}\right)}{\left\| \nabla P_{\bar{a}}\left(z_{\bar{a},x\_n}, z_{\bar{a},y\_n}\right) \right\|} \right\rangle \\ &= \sqrt{\left(x_n - z_{\bar{a},x\_n}\right)^2 + \left(y_n - z_{\bar{a},y\_n}\right)^2} \cdot \begin{cases} 1 & P_{\bar{a}}(x_n, y_n) > 0 \\ (-1) & P_{\bar{a}}(x_n, y_n) < 0 \end{cases} \end{aligned} \tag{A.2}$$

Therefore, we can write:

$$\left(dist\left(Z_{\bar{a}}, (x_n, y_n)\right)\right)^2 = \left(x_n - z_{x-n}\right)^2 + \left(y_n - z_{y-n}\right)^2 \tag{A.3}$$

The goal of the algorithm is the minimization of:

$$E = \sum_{n=1}^{N} dist^2\left(Z_{\bar{a}}, (x_n, y_n)\right) \tag{A.4}$$

In order to achieve this goal, an approximation is made for the distance (first order tailor approximation):

[14] This syntax for the distance expression is useful for finding an approximation for the distance for small errors.

$$
\begin{aligned}
P_{\bar{a}}(x_n, y_n) &\overset{(a)}{\cong} P_{\bar{a}}(z_{x-n}, z_{y-n}) + (x_n - z_{x-n})\frac{\partial P_{\bar{a}}(z_{x-n}, z_{y-n})}{\partial x} + (y_n - z_{y-n})\frac{\partial P_{\bar{a}}(z_{x-n}, z_{y-n})}{\partial y} \\
&\overset{(b)}{=} P_{\bar{a}}(z_{x-n}, z_{y-n}) + \left\langle \left[(x_n - z_{x-n}) \quad (y_n - z_{y-n}) \right] \quad , \quad \nabla P_{\bar{a}}(z_{x-n}, z_{y-n}) \right\rangle \\
&\overset{(c)}{=} P_{\bar{a}}(z_{x-n}, z_{y-n}) + dist(x_n, y_n) \left\| \nabla P_{\bar{a}}(z_{x-n}, z_{y-n}) \right\| \\
&\overset{(d)}{\cong} P_{\bar{a}}(z_{x-n}, z_{y-n}) + dist(x_n, y_n) \left\| \nabla P_{\bar{a}}(x_n, y_n) \right\| \\
&\overset{(e)}{=} dist(x_n, y_n) \left\| \nabla P_{\bar{a}}(x_n, y_n) \right\|
\end{aligned}
\tag{A.5}
$$

Where (a) is the first order tailor approximation; (b) includes a different syntax using inner product; (c) follows from (A.2); (d) holds when the error is small and the gradient remains similar for both the zero-set point and the data point; and (e) results from the fact that the value of the polynomial at any zero-set point is zero.

An approximation for the sum of squared errors can now be calculates using:

$$
E = \sum_{n=1}^{N} dist^2(x_n, y_n) \approx \sum_{n=1}^{N} \left(\frac{P_{\bar{a}}(x_n, y_n)}{\left\| \nabla P_{\bar{a}}(x_n, y_n) \right\|} \right)^2 \tag{A.6}
$$

The first attempt to minimize this expression where made by Taubin, at Brown university. Initially, the minimization problem was solved using non-linear least-squares solutions [26], i.e. gradient descent based algorithms. Such solutions have many local minimas and are sensitive to initialization. Therefore, a reasonably accurate initial value of the coefficient vector was needed to begin the optimization. This initialization was obtained by optimizing an approximated error criteria $\widetilde{E}$:

$$
\widetilde{E} = \frac{\sum_{n=1}^{N} \left(\bar{a}\, \bar{p}^T(x_n, y_n) \right)^2}{\sum_{n=1}^{N} \left\| \nabla P_{\bar{a}}(x_n, y_n) \right\|} \tag{A.7}
$$

Taubin's assumption was that a coefficient vector which minimizes $\widetilde{E}$ is close to the coefficient vector which minimizes E and that the gradient descent algorithm, when initialized to that vector, would converge to the global minimum of E.

The term in (A.7) is optimized using the following:

$$
\widetilde{E} = \frac{\bar{a}\left(\sum_{n=1}^{N} \bar{p}^T(x_n, y_n) \bar{p}(x_n, y_n) \right) \bar{a}^T}{\bar{a}\left(\sum_{n=1}^{N} \bar{p}_X^T(x_n, y_n) \bar{p}_X(x_n, y_n) \right) \bar{a}^T + \bar{a}\left(\sum_{n=1}^{N} \bar{p}_Y^T(x_n, y_n) \bar{p}_Y(x_n, y_n) \right) \bar{a}^T} = \frac{\bar{a}\, M\, \bar{a}^T}{\bar{a}\, D\, \bar{a}^T} \tag{A.8}
$$

where

$$M = \sum_{n=1}^{N} \overline{p}^{T}(x_n, y_n)\overline{p}(x_n, y_n),$$
$$D = \left(\sum_{n=1}^{N} \overline{p}_X^{T}(x_n, y_n)\overline{p}_X(x_n, y_n) \right) + \left(\sum_{n=1}^{N} \overline{p}_Y^{T}(x_n, y_n)\overline{p}_Y(x_n, y_n) \right) \tag{A.9}$$

and

$$\overline{p}_X(x_n, y_n) = \frac{d}{dx}\overline{p}(x_n, y_n),$$
$$\overline{p}_Y(x_n, y_n) = \frac{d}{dy}\overline{p}(x_n, y_n) \tag{A.10}$$

The expression in (A.8) is minimized using generalized eigenvectors [4,27,28], and the value obtained serves as initialization for the non-linear least squares (gradient descent) algorithm used to solve (A.6).

This page intentionally left blank

Appendix B – Complexity of exhaustive search for segmentation

This appendix derives an expression for the complexity involved in performing an exhaustive search for the optimal curve segmentation.

In section 4.3.1 we concluded that 2^{Np} possible curve segmentations exist for closed curves where Np is the number of data points. However, each segment may be described using an IP of a different order. Therefore, for a given curve segmentation, there exist MAX_ORDER^{Nseg} combinations of polynomial at different orders where MAX_ORDER is the maximal allowed order of the polynomials and N_{SEG} is the number of segments representing the curve.

Recall the chain equivalent to the curve, with elements being either connected or disconnected, representing points belonging to the same or to different segments. Each segmentation would be indexed by a binary number with Np bits, where each '1' bit marks the beginning of a new segment, and each '0' bit marks the continuity of a segment. The number of segments of the segmentation with index i would therefore be the number of '1' bits in the binary number i. We denote that number using the expression: $\#BITs(i)=1$.

Therefore, the overall number of combination for representing all possible segmentations with polynomial of all possible orders is $\sum_{i=0}^{2^{Np}} (MAX\_ORDER)^{\#BITs(i)=1}$. This expression evaluates to $\sum_{i=1}^{Np} (MAX\_ORDER)^{i} \binom{i}{Np}$. The same applies to open curves with the number of points changing to $Np-1$.

This calculation is not intended as a guideline for implementing an exhaustive search algorithm, but rather to emphasize the complexity of the curve segmentation problem.

This page intentionally left blank

Appendix C – Selection of constraint points

This appendix describes the selection of constraint points used for the constrained least squares fitting described in section 5.1.

Internal and external stripes (S_{INT}, S_{EXT}) are constructed around the fitted curve. Since the points on the curve are normalized to the range of $[-1,1]$ in both horizontal and vertical axes, we define the stripe relative to the normalized data.

The value of the functions (S_{INT}, S_{EXT}) is "1" inside each stripe and "0" outside.

The distance between the curve and the constraint strip is $d_1 = 0.05$. The thickness of the strip is $d_2 = 0.07$ (see Fig C.1). Internal and external constraint points (C_{INT}, C_{EXT}) are sampled inside of the constraint strip according to the following:

$$\begin{aligned} &m = 1,...,40 \\ &l = 1,...,40 \\ &(x_m, y_l) \propto [-0.025, 0.025] - uniform\, distributin \\ &C_{INT|EXT} = \{(x_m, y_l) : S_{INT|EXT}(x_m + 0.05m, y_l + 0.05l) = 1\} \end{aligned} \tag{C.1}$$

These constants are a result of empirical tests. The sampling scheme, which includes random sampling points is intended to eliminate spurious zero-set regions from the constraint stripes.

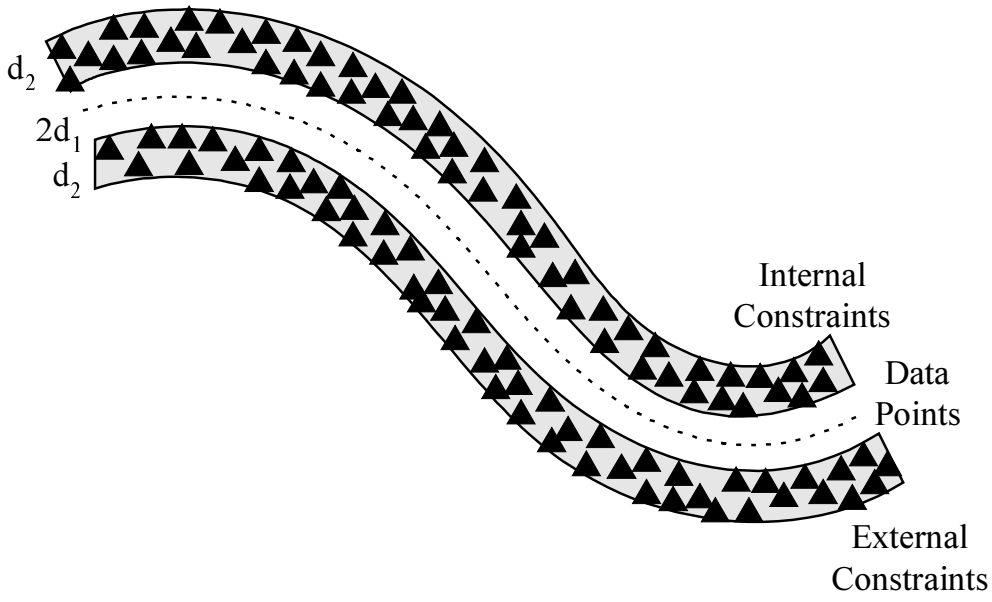


Fig C.1: Selection of constraint points

This page intentionally left blank

Appendix D – Steepest descent algorithm

The steepest descent algorithm is an iterative algorithm for solving non-linear equations.

For the problem of finding an n[th] dimensional vector $\bar{x} = [x_1 \quad ... \quad x \quad]$, for which the function $P(\bar{x}) : n \to \mathbf{R}$ returns zero, we wish to find a solution of the form:

$$\begin{aligned} &\bar{x}(0) = \bar{x}_0 \\ &\bar{x}(n+1) = \bar{x}(n) + l(n)\frac{\dfrac{dP(\bar{x}(n))}{d\bar{x}}}{\nabla P(\bar{x}(n))} \quad n \geq 1 \end{aligned} \tag{D.1}$$

Each step is performed in the direction of the gradient. For infinitesimally small steps this algorithm converges to the nearest local minimum point of $P(\bar{x})$. This algorithm sets the step size between each iteration – $l(n)$ – so that the fastest convergence is achieved.

The first order Taylor approximation of $P(\bar{x})$ is:

$$P(\bar{x}_z + \bar{\varepsilon}) = P(\bar{x}_z) + \bar{\varepsilon}\frac{dP(\bar{x}_z)}{d\bar{x}} \tag{D.2}$$

When $P(\bar{x}_z) = 0$, and $\bar{\varepsilon} = \varepsilon\dfrac{\dfrac{dP(\bar{x}_z)}{d\bar{x}}}{\nabla P(\bar{x}_z)}$ – i.e., points in the direction of the gradient and has a magnitude of e, then (D.2) can be reduced to:

$$P(\bar{x}_z + \bar{\varepsilon}) = \varepsilon\left\|\frac{dP(\bar{x}_z)}{d\bar{x}}\right\| = \varepsilon \cdot \nabla P(\bar{x}_z) \tag{D.3}$$

Neglecting all terms but the 1[st] order derivatives $\bar{x}(n+1)$ is obtained by:

$$\bar{x}(n+1) = \bar{x}(n) - \frac{P(\bar{x}(n))}{\nabla P(\bar{x}(n))} \cdot \frac{\dfrac{dP(\bar{x}(n))}{d\bar{x}}}{\nabla P(\bar{x}(n))} = \frac{P(\bar{x}(n))}{(\nabla P(\bar{x}(n)))^2}\frac{dP(\bar{x}(n))}{d\bar{x}} \tag{D.4}$$

The step size (l) in (D.1) is therefore set to:

$$l(n) = \frac{P(\bar{x}(n))}{\nabla P(\bar{x}(n))} \tag{D.5}$$

This page intentionally left blank

References

[1] T.W. Sederberg and D.C. Anderson, “Implicit Representation of Parametric Curves and Surfaces”, Computer Vision, Graphics, and Image Processing, Vol.28, No.1, pp. 72-84, 1984.

[2] D. Forsyth, J.L. Mundy, A. Zisserman, C. Coelho, A. Heller, and C. Rothwell, “Invariant Descriptors for 3D Object Recognition and Pose”, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 13, No. 10, pp. 971-992, Oct. 1991.

[3] D.A. Forsyth, “Recognizing Algebraic Surfaces from Their Outlines”, Proc. Int'l. Conf. Computer Vision, pp. 476, 480, Berlin, May 1993.

[4] G. Taubin, “Estimation of Planar Curves, Surfaces and Nonplanar Space Curves Defined by Implicit Equations, with Applications to Edge and Range Image Segmentation”, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 13, No. 11, pp. 1115-1138, Nov. 1991.

[5] G. Taubin, F. Cukierman, S. Sullivan, J. Ponce, and D.J. Kriegman, “Parameterized Families of Polynomials for Bounded Algebraic Curve and Surface Fitting”, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 16, pp. 287-303, 1994.

[6] D. Keren, D. Cooper, and J. Subrahmonia, “Describing Complicated Objects by Implicit Polynomials”, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.16, No.1, pp. 38-53, Jan. 1994

[7] Z. Lei, M.M. Blane, and D.B. Cooper, “3L Fitting of Higher Degree Implicit Polynomials”, Technical Report LEMS TR-160, Brown University LEMS Lab., 1997.

[8] Z. Lei and D.B. Cooper, “New, Faster, More Controlled Fitting of Implicit Polynomial 2D Curves and 3D Surfaces to Data”, IEEE Conference on Computer vision and Pattern Recognition, (San Francisco), June 1996.

[9] Z. Lei and D.B. Cooper, “Linear Programming Fitting of Implicit Polynomials”, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 20, No. 2, pp. 212-217, Feb. 1998.

[10] M.M. Blane, Z. Lei, H. Civil and D.B. Cooper “The 3L Algorithm for Fitting Implicit Polynomials Curves and Surface to Data”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 3, March 2000.

[11] D. Keren, "Using Symbolic computation to Find Algebraic Invariants", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 16, pp. 1143-1149, 1994.

[12] J. Subrahmonia, D. Cooper, and D. Keren, "Practical Reliable Bayesian Recognition of 2D and 3D Objects Using Implicit Polynomials and Algebraic Invariants," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 18, pp. 505-519, 1996.

[13] M. Barzohar, D. Keren, and D. Cooper “Recognizing Groups of Curves Based on New Affine Mutual Geometric Invariants, with Applications to Recognizing Intersecting Roads in Aerial Images” IAPR International Conference on Pattern Recognition, (Jerusalem, Israel), October 1994.

[14] D.J. Kriegman and J. Ponce, "On Recognizing and Positioning Curved 3D Objects from Image Contours," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, pp. 1127-1137, December 1990.

[15] K. Siddiqi, J. Subrahmonia, D.B. Cooper, and B.B. Kimia, "Part-Based Bayesian Recognition Using Implicit Polynomial Invariants," IEEE International Conference on Image Processing, (Washington, D.C.), October 1995.

[16] Y. Zhao “Improved Segmentation and Extrapolation for Block-Based Shape-Adaptive Image Coding”, M.Sc. Thesis, Department of Electrical Engineering, Technion, Israel, July 1999.

[17] Y. Zhao and D. Malah “Improved Segmentation and Extrapolation for Block-Based Shape-Adaptive Image Coding”, pp. 388-394, Proceedings Vision Interface ‘2000, Quebec.

[18] S. Sullivan, L. Sandford, and J. Ponce, “Using Geometric Distance Fits for 3D Object Modeling and Recognition” IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 16, pp. 1183-1196, 1994.

[19] D. Keren and C. Gotsman “Fitting Curves and Surfaces With Constrained Implicit Polynomials” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 21, No. 1, January 1999.

[20] G. Schuster, G. Melnikov, and A. Katsaggelos "Operationally Optimal Vertex-Based Shape Coding", IEEE Signal Processing Magazine, November 1998.

[21] L. Torres and M. Kunt "Video Coding – the second generation approach", Kluwer Academic Publishers, 1996.

[22] A.K. Jain "Fundamentals of Digital Image Processing", Prentice Hall Information and Systems Sciences Series, 1989.

[23] P.A Chou, T. Lookabaugh and R.M. Gray "Entropy-constrained vector quantization", IEEE Transaction on Acoustics Speech and Signal Processing, pp. 31-42, January 1989.

[24] A. Gersho and R.M. Gray "Vector Quantization and Signal Compression", Kluwer Academic Publishers, 1992.

[25] T.M. Cover, J.A. Thomas "Elements of Information Theory", Wiley Series in Telecommunication, 1991.

[26] J.E. Dennis and R.B. Shnabel "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall, 1983.

[27] G.H. Golub and C.F. Van Loan "Matrix Computations", John Hopkins University Press, 1983.

[28] G.H. Golub and R. Underwood "Stationary value of the ratio of quadratic forms subject to linear constraints", Z. Angew Math. Phys., 21:318-326, 1970.