Segmentation and Line Filling of 2D Shapes

by

Ana Laura Pérez Rocha

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

© Ana Laura Pérez Rocha, Ottawa, Canada, 2013

Abstract

The evolution of technology in the textile industry reached the design of embroidery patterns for machine embroidery. In order to create quality designs the shapes to be embroidered need to be segmented into regions that define different parts. One of the objectives of our research is to develop a method to automatically segment the shapes and by doing so making the process faster and easier. Shape analysis is necessary to find a suitable method for this purpose. It includes the study of different ways to represent shapes. In this thesis we focus on shape representation through its skeleton. We make use of a shape's skeleton and the shape's boundary through the so-called feature transform to decide how to segment a shape and where to place the segment boundaries. The direction of stitches is another important specification in an embroidery design. We develop a technique to select the stitch orientation by defining direction lines using the skeleton curves and information from the boundary. We compute the intersections of segment boundaries and direction lines with the shape boundary for the final definition of the direction line segments. We demonstrate that our shape segmentation technique and the automatic placement of direction lines produce sufficient constrains for automated embroidery designs. We show examples for lettering, basic shapes, as well as simple and complex logos.

Acknowledgements

First, I would like to thank Dr. Jochen Lang for all his support, teachings and encouragement during my studies and his kind guidance throughout my research.

Second, I would like to thank our industrial partner for all their support and feedback during the development of this research project.

Also I would like to thank my colleague Andres Solis for his support regarding the skeletal representation used in our research.

Last but not least, I want to thank my family for giving me the opportunity to continue with my studies and for all their support, understanding and encouragement.

Contents

1 Introduction **1**

1.1 Problem definition 2

1.2 Background 2

1.3 Thesis Statement 4

1.4 Thesis Overview 5

1.5 Contributions 6

2 Related Work **7**

2.1 Shape Skeleton 7

2.1.1 Skeletonization Techniques 8

2.1.2 Grassfire Simulation 9

2.1.3 Thinning Methods 10

2.1.4 Skeleton based on Voronoi Diagrams 11

2.1.5 Skeleton based on Distance Transform 12

2.1.6 Properties 14

2.1.7 Applications 16

2.2 Non-Photorealistic Rendering (NPR) 18

2.2.1 Illustrations and Direction Fields 19

2.2.2 Simulation of Textures 21

2.2.3 NPR in Embroidery 22

2.3 Shape Segmentation 22

2.3.1 Definition 22

2.3.2 Skeleton-based Shape Segmentation 23

2.4 Summary 25

3 Skeletonization and Shape Analysis **27**

3.1 Shape Skeleton 27

3.1.1 Feature Transform (FT) 28
3.1.2 Integer Medial Axis (IMA) 30
3.1.3 Contour Approximation 31
3.1.4 Skeleton Computation 33
3.1.5 Skeleton Pruning 34
3.1.6 Vector Representation of the Skeleton 35
3.2 Skeleton Analysis 37
3.2.1 Point-on-the-Curve Algorithm 37
3.2.2 Boundary-Skeleton Mapping 40
3.3 Summary 42

4 Shape Segmentation **44**
4.1 Linear Segment Boundaries 44
4.2 Placement of Segment Boundaries 45
4.2.1 Computation of Segment Boundaries Lines 47
4.2.2 Line-Cubic Bezier Curve Intersection 51
4.3 Special Cases 53
4.3.1 Extra Pruning 53
4.3.2 Multiple Branch Points 57
4.4 Intersections 62
4.4.1 Closest Curve 63
4.5 Discussion 64
4.6 Summary 65

5 Direction Lines **67**
5.1 Direction Lines 67
5.2 Direction Lines Placement 68
5.2.1 Midpoints Computation 68
5.2.2 De Casteljau's Algorithm 70
5.2.3 Obtaining Direction Lines 73
5.3 Direction Lines at Segment Boundaries 75
5.4 Special Cases 76
5.5 Intersections 80
5.5.1 Line-Line Intersection 82
5.5.2 Results 84
5.6 Summary 86

6 Results **87**
6.1 Lettering 90
6.2 Special Shapes 102
6.3 Basic Logos and Figures 105
6.4 Complex logos 110
6.5 Summary 117

7 Conclusions and Future Work **118**
7.1 Summary 118
7.2 Limitations 119
7.3 Future work 120

List of Tables

6.1 Processing time by category 89
6.2 List of observed errors during the processing of embroidery designs 89

List of Figures

1.1 Examples of machine embroidery designs. 3
1.2 Thesis Overview. 4

2.1 Medial Axis of a regular shape 9
2.2 Voronoi Diagram-based Skeleton. 12
2.3 Example of an object and its medial axis. 16
2.4 Local geometry of skeletal points. 17
2.5 Overview of the system for pen-and-ink illustrations. 21

3.1 Algorithm to compute the Feature Transform, first stage 29
3.2 Feature Transform, second stage. 30
3.3 IMA points selection. 31
3.4 Contour Approximation. 33
3.5 Selection of Skeleton points. 34
3.6 Procedure to identify skeleton points. 35
3.7 Branch removal according to Equation 3.5. 36
3.8 Skeletal Representation Example. 36
3.9 Point-on-the-Curve. 38
3.10 Root-finding algorithm. Adapted from [19]. 40
3.11 Skeleton-boundary mapping. 41
3.12 Example of mouse interaction. 42
3.13 Relation between the skeleton and the boundary curves. 43

4.1 Example of curves merging. 45
4.2 Branch point and the feature transform of its neighboring pixels. 46
4.3 Algorithm to compute Segmenting Lines. 46
4.4 Shape elements used for segment boundary computation. 47
4.5 Shape and Segment Boundaries (red). 48

4.6 Pseudocode for $getSL(\ )$ method, part (a). 49
4.7 Pseudocode for $getSL(\ )$ method, part (b). 50
4.8 Example of a cubic Bezier curve. 51
4.9 Skeleton-segment boundary intersection algorithm. 54
4.10 Example of shapes with an inaccurate segmentation. 55
4.11 Example of squared-ends shape pruning. 55
4.12 Example of shapes with their segment boundaries after pruning. 56
4.13 Pruning step exception. 56
4.14 Branches removed after changing pruning threshold. 57
4.15 Shape with branch point joined by a short curve segment. 58
4.16 The star shape has all its branch points right next to each other. 58
4.17 Segmentation using the general method. 59
4.18 Maximal disks and branch points. 59
4.19 Segmentation for shapes with bp close to each other. 60
4.20 Results after removing some segment boundaries. 60
4.21 Example of shape with multiple branch points. 61
4.22 Algorithm to compute intersections. 62
4.23 Algorithm to find the closest curve to an arbitrary point. 63
4.24 Intersections by using k-d trees. 64
4.25 Segmentation results. 65

5.1 Process for the direction lines computation. 68
5.2 Algorithm to compute direction lines. 69
5.3 Midpoints computation (a). 71
5.4 Midpoints computation (b). 72
5.5 Curve subdivision by De Casteljau's algorithm. 73
5.6 Obtaining the preliminary direction line segments for t=0.5. 74
5.7 Obtaining longer direction line segments. 75
5.8 Algorithm to obtain larger preliminary direction lines. 76
5.9 Algorithm to compute the direction line segments at segment boundaries. 77
5.10 Shape with its segment boundaries and direction lines. 78
5.12 Small shape with no skeleton and its direction line segments. 78
5.11 Computation of direction line segments for small shapes. 79
5.13 Segmented shape and direction lines. 80
5.14 Intersections computation. 82

5.15 Method to compute intersections, returns the intersection points. 82
5.16 Lettering Examples 84
5.17 Star shapes with segment boundaries and direction line segments. 85
5.18 uOttawa logo. 85

6.1 Letter P results 90
6.2 Letter Q results 91
6.3 Letter X results 91
6.4 Font Liberation Serif, lower cases 93
6.5 Font Liberation Serif, capital letters 94
6.6 Font Liberation Serif, segment boundaries and direction lines 95
6.7 Font Purisa, lower cases 96
6.8 Font Purisa, capital letters 97
6.9 Font Purisa, segment boundaries and direction lines 98
6.10 Font Eufm, lower cases 99
6.11 Font Eufm, capital letters 100
6.12 Font Eufm, segment boundaries and direction lines 101
6.13 Star shapes, example 1 102
6.14 Star shapes, example 2 103
6.15 Circles 104
6.16 Square 104
6.17 South Korea Flag 106
6.18 Horse 107
6.19 Eye-logo 108
6.20 Bear 109
6.21 Bee 109
6.22 Thanks 112
6.23 Fish 113
6.24 University of Ottawa logo 114
6.25 Brain Skeleton 115
6.26 Brain 116

Chapter 1

Introduction

The use of computer equipment and software is ever increasing and, currently, there is no facet of human activity where computers are not present. One of these activities where computer applications are of great importance is the design field, from architecture and civil engineering to fashion, graphic arts and marketing. Related to fashion and marketing, we find the use of computational methods for the creation of automatic embroidery designs.

Embroidery is considered a type of art where the artists create a design by means of needles and thread on fabric. There exist many techniques or stitch types to produce an embroidery design or stitch pattern. The so-called satin stitch is one of the most popular ones. The satin stitch [40] is by definition a series of parallel flat stitches placed close together to give a satin-like finish. The creation of an embroidery design is an expensive process because of the amount of time required for it, and hence, the increasing demand in the textile industry for mass production of embroidery designs. The evolution of technology has led to the development of design software and computer controlled machines for automated embroidery. One goal in machine embroidery is to stitch an embroidery design that resembles a design done by hand while at the same time making the process of creating such a design easier and faster.

In the automation of embroidery the input or initial design is a 2D image (raster or vector image) from which 2D shapes are extracted. The 2D shapes are then processed resulting in the embroidered design, also called stitch pattern. Image-based techniques and the analysis of the geometric properties of 2D shapes are particularly important in this translation process. In order to produce a quality automated embroidery design, it is necessary to define how a shape will be filled, this is, to determine the direction of the

stitches and what parts of the shape should be embroidered with stitches that have the same direction. Therefore, segmentation techniques need to be implemented to divide a shape into different regions, as well as methods to establish the direction of the stitches to be placed in every region and their flow.

This thesis addresses the above mentioned, the analysis of the structure of 2D shapes and its geometric properties. The boundary of a shape and shape representation such as medial axis or skeleton play an important role in our implementation and we show how our approach can be applied in the automation of embroidery designs.

1.1 Problem definition

The goal of an embroidery design is to give a better or "arresting" look to fabrics or to the materials that contain them, e.g. T-shirts, dresses, caps. Therefore, it is important to keep in mind that embroidery is an artistic process and automated embroidery designs should satisfy the same goals.

When an embroidery design is created by hand, it is easy to identify the different shapes and the different regions of every shape, as well as give the stitches a direction and modify them as needed. Depending on the design, this may be a time-consuming process however, the results are very satisfactory. The above could be considered one of the main differences between hand embroidery and automated embroidery: in the automation, the shapes are not specifically identified. Hence, our prime objective and contribution is the development and implementation of techniques to break a design, in our case a 2D shape, "automatically" into several regions and to place lines (direction lines) on every region to define the orientation of the stitches and how they should be changing along the region in order to create a quality embroidery design. Since we are dealing with irregular shapes, we need to find the best way to segment them and where to place the direction lines without losing focus on the aesthetic aspect. Figure 1.1 shows some examples of machine embroidery designs.

1.2 Background

The focus and purpose of machine embroidery is the decoration of different materials, in particular, to give fabrics a better look. One of the main problems in the automation of embroidery design is the segmentation of shapes into different regions. Properly chosen

Figure 1.1: Examples of machine embroidery designs.

regions produce a quality result. Another major challenge is the placement of lines that control the flow of the stitches during filling of the shapes. To this end we need to study the geometry of 2D shapes closely, particularly the local symmetries which are represented by a shape's medial axis, more commonly known as shape skeleton, which will be briefly described later. Also, since the creation of an embroidery design is an artistic task, we will relate our research to the field of Non-Photorealistic Rendering (NPR) whose aim is to produce computer generated images simulating an artistic style, e.g., pen-and-ink, hatching, stippling and even textures to name but a few.

Shape analysis is essential in systems for shape matching, shape recognition and shape retrieval; and it has many applications in different fields of study e.g. neuroscience, medicine, engineering, biology. It may be divided into three main steps [13]: shape pre-processing, shape transformation and shape classification. Although all the steps are important we will focus on the shape transformation which allows one to obtain information from shapes. More specifically we will take a closer look at the representation and characterization of shapes. The shape representation technique is defined according to the requirements of the implementation which then allows the quantification of some properties. This is the characterization of the shape, where the properties or information obtained from the shape also depend on the problem we need to solve.

There are several techniques to represent a shape. The simplest way is by a series of points. However, sometimes this is not the most practical thing to do due to the amount of data that needs to be stored [13]. In general, these techniques are classified into two groups: boundary-based (or contour-based) techniques [69, 57, 1] and region-based techniques [5, 68]. Contour based techniques include: chain code, polygon approximation, B-spline, Fourier descriptors, autoregressive models, among others. In the region-based group we find techniques such as medial axis, convex hull, shape matrix and grid meth-

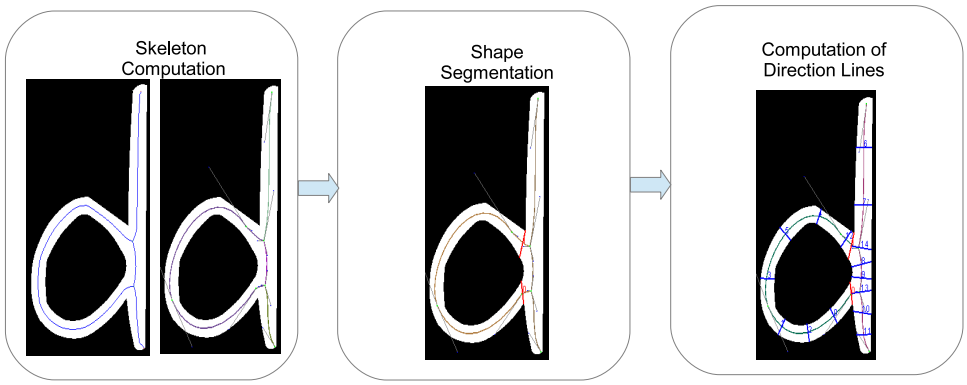


Figure 1.2: System Overview.

ods. Zhang and Lu [70] reviewed and classified these shape representation techniques and their implementation and examine their advantages as well as their drawbacks.

In our research we are interested in representation of a shape by means of its skeleton and then obtain the necessary measures from it. Shape skeletons have been studied for more than 40 years and several techniques for their computation have been developed. De Floriani and Spagnuolo [16] give two possible definitions of a shape skeleton: 1) "An entity that always falls inside the shape and is in each point equidistant from the shape boundary", and, 2) "Explicit representation of how the basic components of the shape are glued together to form a whole". This type of shape representation is very useful in applications where the boundary of the shape needs to be reconstructed. There are a great number of research papers that describe methods to compute a shape skeleton. We will review the most relevant in more detail in Chapter 2 of this thesis.

1.3 Thesis Statement

The automation of embroidery designs requires the development and implementation of algorithms for 2D shape segmentation and filling. Methods to obtain shape descriptors, such as the skeleton, encode important information about the shape that allows us to decide how the shape should be divided into regions and the direction that stitches should have in order to produce embroidery designs from 2D shapes.

1.4 Thesis Overview

Our implementation consists of three main stages as described in Figure 1.2. The first stage is the computation of the input shape (binary image). The skeletal representation we are using in our research was developed and implemented by Solis Montero and Lang [43] and will be described in detail in Chapter 3.

Based on the shape's skeleton, the data obtained from it and the shape's boundary (e.g., feature transform), we devise a technique for shape segmentation. Broadly speaking, in this step we look for two boundary points, each of them at different sides of the skeleton, which are the closest points to a branch point. These two points are used to define a line segment, called segment boundary, which divides the shape into regions.

After shape segmentation, we need to compute the direction lines whose objective is to "control" the flow of the stitches during the embroidery process. Each region obtained from the segmentation contains several skeleton branches and we have to decide how many lines should be placed and the points where the lines will intersect the skeleton curves. The latter is done by means of DeCasteljau's algorithm which also helps us to define preliminary direction line segments which are then extended to ensure they intersect the shape boundary.

Once the segment boundaries and direction lines have been computed, we need to compute the points where they intersect the boundary of the shape in order to define the final lines. Different algorithms have been used for this purpose and they will be described in Chapters 4 and 5.

Finally, we made use of a commercial software for the creation of automatic embroidery designs[1] to test our results and have a better visualization of how the final output (the embroidery design) would look like.

In summary this thesis is organized as follows:

- Chapter 2 contains a review of the literature related to the different methods for computing a shape's skeleton, Non-Photorealistic Rendering and the description of some skeleton-based shape segmentation techniques.

- Chapter 3 describes the method to compute the skeletal representation used in our implementation.

- Chapter 4 describes our method for shape segmentation based on its skeleton.

[1]Software name withheld for confidentiality

- Chapter 5 outlines the steps we follow for the placement of direction lines to give stitches an orientation.
- Chapter 6 provides some of our results for different groups of shapes. We also include the results after processing the shapes as shown inside the software for embroidery design.
- Chapter 7 presents our conclusions and discusses the direction of future work.

1.5 Contributions

The main contributions of our work are:

- Skeleton-based shape segmentation using the information from the feature transform.
- Detection of multiple branch points scenarios to improve the segmentation results.
- Placement of direction lines for their application in an automatic embroidery design systems.

Chapter 2

Related Work

In this chapter, we present a review of the literature related to shape analysis for shape representation. We have focused on the different skeletonization algorithms which are discussed in Section 2.1; we describe important properties of skeletons and their application. In Section 2.2 we provide a brief review of some work in Non-Photorealistic Rendering (NPR), an area of computer graphics and its relationship with the generation of embroidery designs. Shape segmentation based on skeletal representations is discussed in Section 2.3. We conclude this chapter with a summary in Section 2.4.

2.1 Shape Skeleton

Shape analysis plays an important role in applications in many fields of science and engineering. Representation is an essential step in such analysis for the quantification and study of the different properties of the shapes. Representation techniques may be divided into contour-based techniques and region-based techniques depending on where the shape features for the representation were obtained from. In this thesis we will focus on region-based techniques, particularly on those related to the extraction of a shape's skeleton. One of the goals of our research is to break a shape into several regions based on its skeleton in order to decide how a design will be embroidered. In the following we will review some existing work to compute this type of representations, its properties and some applications in different areas of research.

2.1.1 Skeletonization Techniques

There are a great number of publications related to methods for skeletal representation of shapes. These techniques are an important tool in shape analysis that has been studied for more than 40 years. The Medial Axis Transform was introduced by Blum [6] in 1967, using the analogy of a fire front starting along the shape's boundary at the same time propagating in all directions. The points where the fronts meet define what he called *medial axis* or *skeleton*. Then he considered another approach to define the medial axis as the locus of centers of disks that fit inside a shape [7]. We found two more formal definition of the medial axis transform in the literature [10]: Being S a bounded subset of R^n

- The medial axis of S is defined by

  > "The centers of n-dimensional balls that are included in S but that are not included in any other n-dimensional ball in S".

- The medial axis of S

  > "Consists of the points $x \in S$ that have more than one nearest point on the boundary of S".

Figure 2.1 shows the medial axis of a rectangular shape. Obtaining the skeleton of regular shapes may be an easy task and the exact computation of the medial axis is possible, however, this is not the case when we are dealing with shapes of variable form. To solve this problem a medial axis approximation is required. Many techniques have been developed for this purpose and have been classified into four main categories:

1. Grassfire Simulation.
2. Topological Thinning.
3. Skeleton based on Voronoi Diagrams.
4. Skeleton based on Distance Transform.

De Floriani and Spagnulo [16] have reviewed the different skeleton methods and provide a description of each of them discussing several algorithms and their characteristics. Skeletons approximated by Voronoi diagrams and fire fronts simulations are considered continuous methods while thinning and distance transform methods are discrete, this is, the skeleton is represented by a set of connected pixels.

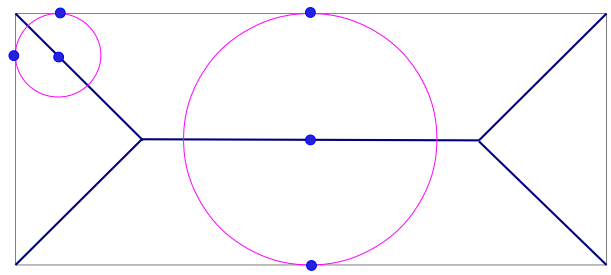


Figure 2.1: Medial Axis of a regular shape

2.1.2 Grassfire Simulation

This type of skeleton approximation is based on the model introduced by Blum [6] where fire fronts propagate inwards from the shape's boundary and the skeleton consists of the set of points where the fronts meet (quench points), as described above. Xia [66] proposes an algorithm to compute the skeleton of binary images by simulating the propagation and extinction of fire fronts. This is done by means of the method known as contour following. In this technique the contour points are identified and marked as well as the points on the next contour or front. That is, the image is scanned row by row assigning values to each contour point and to the points associated to it. The values are: 1, for object's points; 0, for the background; 2 is assigned to the edges. If a point receives a value of 3 or higher, it means that two fronts have met and thus it is a skeleton point. Finally, points with values less than 3 are removed. The process is repeated until no more points can be removed. Another model to simulate the grassfire propagation is the snake, an active contour applied by Laymarie and Levine [33] in their approach for shape representation. A snake is a deformable contour that lies on a surface adopting its shape under the action of a gravitational force. In this case the surface is the distance transform function D(x,y) which controls the movement of the snake. One of the main characteristics of this method is the combination of Blums analogy and a distance transform.

Shah [58] has introduced the gray skeleton where each point of the skeleton is associated with a significance number. In this approach the objective is not the simulation of fire fronts, however, the method is based on its principles. It is necessary to find the values assigned to each point of a shape and be able to decide if a point is part of the skeleton or not. Branches with low values are considered insignificant. The central task of this method is to compute such values which are defined by the angle between the front normals. A pruning method is also proposed to eliminate disconnected branches produced by noise.

In the literature we also find the straight skeleton, Aichholzer et al. [2] present a new technique to compute the skeleton of simple polygons. The method divides the interior of a polygon P of n sides into n internal polygons and the skeleton is represented by a set of straight line segments which are obtained by shrinking P. During the process the edges of P move inwards at the same speed while the topology is preserved, and the vertices define the bisectors which union yields the straight skeleton. Another approach computes the discrete straight skeleton of binary images [14] using polygonal approximation.

2.1.3 Thinning Methods

Thinning is an iterative process where pixels that belong to a certain neighborhood configuration (e.g., a rectangular, triangular or hexagonal grid [47]) are removed from an object's boundary until no more pixels can be deleted while preserving the topology of the shape and leading most of the times to a one-pixel wide skeleton. Another advantage of these methods is they guarantee the connectivity of the final structure. However, one of their drawbacks is that Euclidean metrics are not preserved. Lam et al. [32] discuss a great number of algorithms based on this technique which are classified as sequential and parallel methods. In the sequential algorithms, the pixels are tested in a defined order and the removal of pixels in the current iteration depends on the pixels that have already been tested and the result from the last iteration, whereas in the parallel algorithm the decision of removing a pixel in the current iteration is based only on the result from the previous one. They also considered non-iterative or non-pixel based thinning methods; the result obtained is a center line without going through every pixel on the image. In order to do this, the algorithm carries out a scan in a defined direction looking for intervals of pixels; the skeleton is generated by connecting them. The method proposed by Xia [66], which was described above as a grassfire simulation algorithm, can also be classified as a thinning method, since contour pixels are marked and then deleted according to a neighborhood criteria in an iterative fashion.

Pavlidis [48] has proposed a sequential thinning algorithm by contour tracing which produces a connected skeleton and allows the reconstruction of the original shape. However, in some cases the skeleton may not be centered. Another approach that uses the contour following technique to decide which pixels should be deleted is by Arcelli [3]. It reduces the computation time for the skeleton.

Naouai et al. [44] have developed a new skeletonization technique for binary images

that uses thinning methods as first step in order to obtain the skeleton from the original image. Then the resulting skeleton is pruned to remove insignificant branches. Finally, the pruned structure is approximated by Non Uniform Rational B-Splines (NURBS).

2.1.4 Skeleton based on Voronoi Diagrams

Another technique to compute a shape's skeleton is by means of Voronoi diagrams. The general idea behind this method is to take sample points from the boundary and compute their Voronoi diagram. The Voronoi edges inside the shape represent the skeleton and the exact shape representation can be achieved as the sampling density goes to infinity (see Figure 2.2). Two of the main properties of these methods are the preservation of connectivity and Euclidean metrics [37] which allows the reconstruction of the original shape. Moreover, there exist a close relation between the medial axis and this approximation because both of them require one to compute the minimum distance between points [8].

One algorithm to compute a continuous skeleton applying Voronoi diagrams was proposed by Brandt and Algazi [8]. The first step of the algorithm is to find a sufficiently dense sample of points from the boundary in order to obtain an approximation as close as possible to the continuous skeleton. Then the Voronoi diagram of the sampling is computed and finally, a pruning step is performed. Besides this contribution, the algorithm can be applied to both simple polygons and general shapes (e.g., binary images).

Another approach was developed by Mayya and Rajan [38]. The algorithm here described computes the skeleton of polygons with or without holes. According to the authors most of shapes are "non-polygonal", therefore, a polygonal approximation of the input shape is required. Once this has been done, the Voronoi diagram of the new polygon is computed, however in this case the sites considered for the computation are not only points but lines and the edges in this kind of diagram are straight lines and arc curves. The description to apply this method to polygons with holes is presented in [37].

Ogniewicz and Kubler [47] present another algorithm to obtain the skeleton by computing the Voronoi diagram of a set of points sampled from the boundary. Each edge of the diagram is weighted according to its importance and stability and weights are used as criteria for the removal of branches which leads to the approximation of the continuous Medial Axis Transform. The skeleton obtained by this method is robust under geometric transformations.

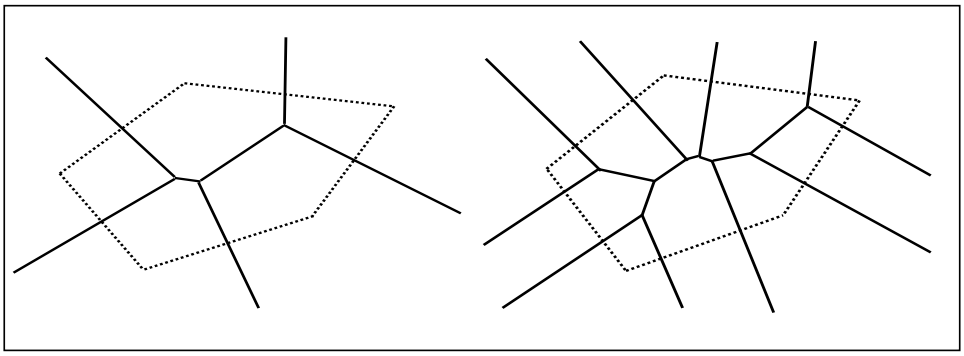


Figure 2.2: Voronoi Diagram-based Skeleton. As the sampling density increases, the Voronoi diagram of the samples approximates the exact skeletal representation of a shape [8].

2.1.5 Skeleton based on Distance Transform

It can be said that this kind of skeletons are the closest approach to Blum's medial axis definition. In these methods each point of the shape is associated with the distance to the nearest point on the boundary. There exist different distance map functions (e.g. Euclidean, Manhattan distance, Chessboard distance) and the resulting skeleton depends on the applied function [16].

One of the disadvantages is that the simplest algorithms do not use the Euclidean distance and the result is not reliable regarding Blum's analogy and those where it is applied yield a disconnected skeleton [37]. Also these methods may present a skeleton sensitive to rotations and noise. Before reviewing some methods based on this approximation technique we need to define the distance transform or distance map. According to Ogniewicz and Kubler [47] a distance map can be defined as follows: "Given object S, for each $(x, y) \in S$, we compute the associated distance value $\rho(x, y)$ with respect to a distance function d according to

$$\forall (x, y) \in S : \rho(x, y) = \min_{(x_t, y_t) \in \overline{S}} d[(x, y), (x_i, y_i)] \tag{2.1}$$

The set of all $\rho(x, y)$ is called the distance map DM(S) of S".

Niblack et al. [45] have developed an algorithm to compute the skeleton of binary images focusing their effort on achieving three important properties: the resulting skeleton should be connected, centered and thin. The reconstruction of the original shape is possible within a margin of error or tolerance. The skeleton is not always thin, therefore

a thinning method is also presented, however, this affects the reconstruction step and it is necessary to minimize the error during the reconstruction. The distance map used in this case is the so called "chamfer 2-3" and "chamfer 3-4". This type of metric, for example chamfer 2-3 gives a distance value of 2 if neighboring pixels are in the horizontal or vertical direction and a distance of 3 to pixels in diagonal. The distance transform of a pixel p follows the definition described above: " $d(p,q) = \min[d(p,q)]|q$ does not belong to any object ".

Malandain et al. [36] devised a method to compute the skeleton of 2D shapes by means of the Euclidean distance which allows centered skeleton. Two parameters are used to control the details of the output and must be chosen carefully since a low threshold preserves the topology of the shape but the skeleton is sensitive to noise. On the other hand a large threshold results in a disconnected skeleton. As well as in the previous method, a reconstruction algorithm which helps to preserve the connectivity and reduces the sensitivity of the skeleton to noise is presented.

Now we will describe the λ-medial axis by Chazal and Lieutier [11] which is described as a subset of the medial axis. The points that form the λ- medial axis are defined by the disks that contain the closest points on the boundary and have $radius \geq \lambda$. This representation can be expressed [4]:

$$M\lambda[X] = x \in X|r(x) \geq \lambda \tag{2.2}$$

In this algorithm the boundary of the shape is approximated by the Hausdorff distance which according to the authors is defined as follows:

> "Let A and B be two compact subsets of R^n. The Hausdorff distance between A and B is defined by
>
> $$dH(A,B) = \max(\sup_{x \in A} d(x,B), \sup_{y \in B} d(y,A))".$$

The function *sup* can be defined as follows: assume a set of upper bounds of $d(x,B)$, the smallest element of such a set is the supremum (sup) of $d(x,B)$.

It has been proven that the skeleton remains stable under small changes in the Hausdorff distance. Another concept in this method is the weak feature size (wfs) which is the minimum distance between the points on the boundary and the critical points in the object. The critical points p are those ones with $\nabla(p) = 0$. When $\lambda < wfs$ the medial axis preserves its connectivity. A discrete approach of the λ- medial axis was presented

by Chaussard et al. [10]. The shape representation obtained with this algorithm is robust under the presence of noise and rotations. A set of points in Z^n is defined by X and being λ a positive real number, for each point p in X, a neighborhood N(p) is defined by means of the Euclidean distance. The function F(p) is equal to radius of the smallest disk that includes the points in N(p). Then the discrete λ-medial axis of X is formed by the points in X with $F(p) \geq \lambda$.

Hesselink and Roerdink [28] have used the Euclidean distance in their method to compute the skeleton of 2D and 3D shapes which they call Integer Medial Axis (IMA) using the computation of the Feature Transform as the first step of the algorithm. The IMA is defined as the set of closest points on the boundary for each point in the shape. That is, the feature transform not only associates every point in the shape with the minimum distance to the boundary but also with a point at such a distance. The skeleton of the shape is then computed based on the feature transform. Three pruning methods: constant, linear and square-root pruning are another contribution presented in this paper. One of the disadvantages of this method is a potentially disconnected skeleton. This algorithm is particularly important in our research since the skeletal representation we are using in our implementation is based on the Integer Medial Axis transform of Hesselink and Roerdink. We will describe the mentioned skeleton in detail in the next chapter of this thesis.

In their work, Giesen et al. [18] have presented the Scale Axis Transform which is based on the medial axis transform defined by the union of medial disks. They use a multiplicative scaling to measure the importance of the shape features eliminating the small ones. The scale axis transform is computed by first obtaining the Medial Axis of the original shape. Then the shape is approximated scaling the medial disks by a factor s. The scale axis transform is the medial axis transform of the approximated shape. Finally the shape is scaled back by a factor $1/s$. This approach is extended in [42] to compute a simplified representation of 3D objects, polygonal meshes and sets of points.

2.1.6 Properties

We have reviewed so far the techniques and some algorithms available in the literature to compute a shape's skeleton, in this section we will described briefly some of the most important characteristics this structure should have. Recall that the medial axis has been defined as the set of points that have two or more closest points in the shape's boundary [10] and it is considered as an abstract representation [61] of the original

shape. It provides us with significant information about the shape since it uses data from the boundary and associates it with data from inside the shape, e.g., the radii of the medial disks that relate each medial point with its distance to the boundary [50]. These important characteristics have made the skeleton an important tool in many applications in computer graphics, computer vision, image analysis, to name but a few and which we will discuss in the next section.

Three of the most important properties of a skeleton [36, 45] are: The skeleton should have a homotopic equivalence with the object, that is the object and its skeleton hold the same connectivity. Also, it should be invariant in the presence of geometric transformations such as rotations and as mentioned above, when the skeleton is associated to a radius function [50] it should allow the reconstruction of the original shape. The skeleton should be thin and if it is based on a distance map it should be centered in the shape. Besides these desirable properties, it is noted that skeletons present instability and are sensitive to noise generating unwanted branches. However, this is not always an inconvenience because it is the application which dictates the importance of the skeleton properties.

Geometrically speaking we find different types of points in a skeletal representation which are classified into three categories: end points, normal points and branch points, depending on the number of adjacent skeletal points they have, 1, 2, 3 or more, respectively. This also can be defined by the order or degree of the point, taking into account the number of times a disc centered at each point intersects the skeleton, therefore a point can be of 1st, 2nd or 3rd or higher degree [35]. Another way of assigning a degree to a point is by the number of times a medial disk touches the boundary [62]. In this case a point on a disk can touch a single point on the boundary or a set of points of the disk forming an arc can touch the boundary. See Figure 2.3, where a simple shape and its medial axis are shown with the different types of points, their medial disks and intersections with the skeletal branches and their degree, as well as the touching points with the boundary. Siddiqi et al. [60] provide a detailed description of the local geometric properties of the medial axis considering the normal points of the skeleton. Each point has a position p and a radius r. The vectors formed by the two boundary points touched by the disk of radius r and the medial point are named spokes by Siddiqi et al. The object's angle is defined by the spokes and the unit tangent vector U (Figure 2.4).

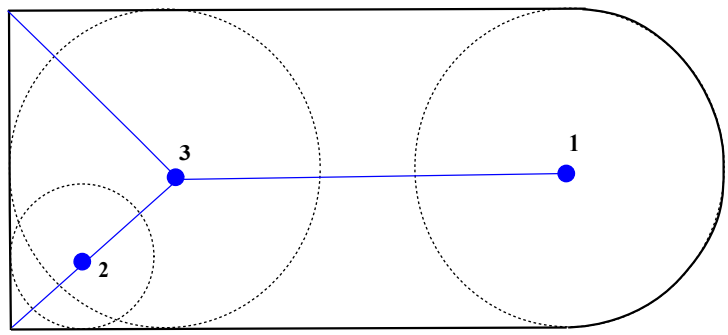


Figure 2.3: Object and its medial axis defined by the skeletal branches. The numbers indicate the degree of the points and the circles represent the maximal disks at three different skeleton points. Adapted from [62].

2.1.7 Applications

As we mentioned before, the skeleton or medial axis is a shape representation that, because of its properties and the object's features that one can obtain from it, has advantages in many applications in different fields of science and engineering, e.g., biology, neuroscience, and computer vision. Most of these applications are related to shape matching, object recognition and classification. This section describes some of them in more detail.

Blum [7] applies his medial axis concept to the representation of shapes in Biology, from cells to complex organisms and relates it with the human vision to understand how we perceive and describe them. This research motivated the shape representation approach of Laymarie et al. [33] (already described) while Malandain et al. [36] found the application of their skeleton in the field of 3D images in Medicine to represent the corrugation of the surface of the brain.

Torsello et al. [61] use the features extracted from the skeletal representation of 2D shapes to measure and differentiate between similar shapes. Their approach also allows identifying the ligature and non-ligature points in the skeleton. Macrini et al. [35] make use of branch points and the skeleton's ligatures which are defined by the protrusions in the boundary, to decide what branches could be merged and what junctions could be removed in order to define an skeleton based on bones and ligaments. Here, the bones represent a mapping between shape parts and skeleton branches in a one-to-one fashion and ligaments are the connections between bones. This yields a bone graph which has applications in object recognition tasks.

Different techniques to represent shapes by their medial axis and how to apply these

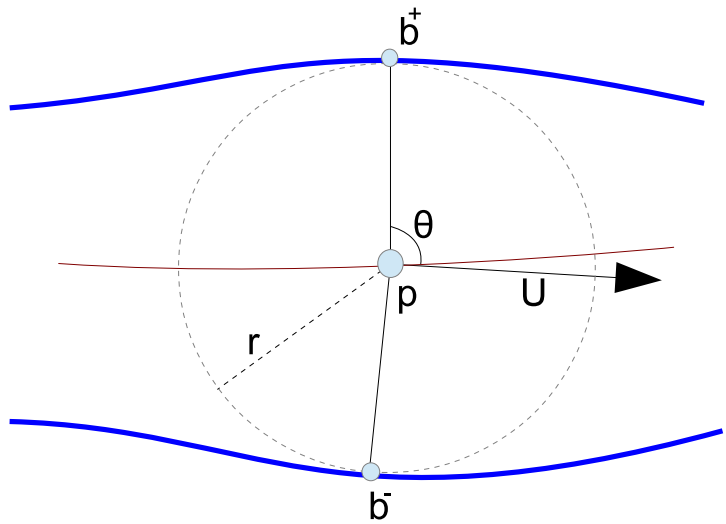


Figure 2.4: Local geometry of skeletal points, b^+ and b^- represent the two points touched by the disk with radius r centered at p. The angle θ is defined by one of the spokes vector and the unit tangent vector U. Adapted from [60].

representations to CAD systems for feature recognition in both 2D and 3D objects have been also studied [50]. This and other applications are described by Quadros et al. [51] where the authors present the use of the medial axis in the manufacturing industry. Another application in engineering was provided by Niblack et al. [45]; their skeleton algorithm by means of distance transform has characteristics that makes it helpful in some areas of Electrical Engineering, e.g., examination of printed circuit boards.

In 2008, Siddiqi et al. [60] provide a detailed description of the mathematics behind the medial axis, e.g., its geometric properties. They also provide an explanation of the different algorithms to compute it, such as distance transform methods for discrete skeletons in 2D and 3D as well as the Voronoi methods to approximate a 3D medial axis. The conclusion of their book (Chapter 11) gives a great number of examples of the different areas where the medial axis is applied, these areas include geography where the skeleton has been used in topography and cartography; civil engineering, architecture and archaeology in applications where the representation of 3D objects is important for Geographical Information Systems (GIS) and feature description of fragments found in archaeological expeditions for object matching. Non-Photorealistic rendering, a branch of Computer Graphics where the medial axis has been applied to determine brush strokes, its inclination and length. Another application in this field is the simulation of the move-

ment of articulated objects and animation to name but a few. Some other applications such as segmentation of shapes based on their medial representation are of particular interest in our research, therefore we will review the literature related to this topic in Section 2.3

2.2 Non-Photorealistic Rendering (NPR)

The concept of rendering is referred to as the process of translating mathematical data into an image [23]. Photorealistic rendering has been part of the field of computer graphics for more than 50 years. The goal in computer graphics is often to create images that look just like photographs or, in virtual reality (VR), to create images that resemble the real world. Pomaska [49] mentioned in his work that in some cases one can even use the term ultra realistic rendering to describe high quality images applied in VR. However, because of the level of detail of this kind of images, the communication of concepts and ideas is not always optimal since the quality of photorealistic rendering is measured by how similar they are to reality and by how much content they possess.

In contrast, Non-Photorealistic Rendering or NPR is a relatively young area in computer graphics which can be defined as the series of "algorithms that imitate non-photographic techniques" [22]. These techniques refer mainly to artistic styles. The simulation includes pen-and-ink illustrations, painterly rendering, cartoons, technical illustrations, among others. NPR often is done by means of image and geometric processing techniques. As described by Gooch et al. in [24, 23], unlike photorealistic rendering, in NPR, the quality of an image is defined by how it communicates ideas. Non-photorealistic images are a convenient way of focusing on a better understanding of the illustrated concept. This can be achieved changing the "level of abstraction" of an image, depending on what is to be transmitted and centering attention on important characteristics.

As described above, NPR allows obtaining an abstract representation of images simulating artistic styles. Based on this definition we can relate our work to NPR in the sense that we want to represent an image by means of an embroidery design. Although we are mainly interested in designs with satin-stitches, different stitching styles may be used to give the designs different textures. In either case, the stitches need to have certain angle or direction in order show different stitching styles and produce satisfying results. For these reasons we will describe some NPR techniques related to our research, e.g., simulation of textures, NPR techniques that make use of direction fields, and application

of NPR concepts in the context of embroidery designs.

2.2.1 Illustrations and Direction Fields

Illustrations represent a useful tool to transfer information based on abstracted images which are widely used in printed media because they present the ideas in a clearer way. One of most important method to create illustrations is the pen-and-ink method. Pen-and-ink illustrations are created by individual pen strokes. Although, it is considered an economical and simple method, certain level of knowledge and skills are required to generate the illustrations. A single stroke lacks of expression, however by combining them it is possible give the illustrations a texture and tone. When it comes to textures it is important to take the stroke direction into account. Some of the most popular textures used in pen-and ink illustrations are: hatching, and stippling. The density of the strokes gives rise to tones [55]. Despite their "simplicity", pen-and-ink illustrations require a substantial creation and if a mistake is made there is no way to correct it. It could be said that this situation has been a motivation for the development of computer systems that create pen-and-ink-like illustrations in an easier and faster way.

Salisbury et al. [55] have designed a mouse-based interactive system to generate pen-and-ink illustrations. The system was provided with different stroke textures to choose from allowing the user the creation of illustrations with different styles. In their work, Winkenbach et al. [64] also provided a detail description about pen-and-ink illustrations and present a computer system to create them. Their implementation allows the user to place what they called detail segments to let the system know where to apply texture details in the image. They also introduced the concept of prioritized stroke texture to give tone to the illustrations. Strokes of highest priority are placed as first layer therefore, vertical strokes are placed first and then the horizontal ones which in turn have higher priority than diagonal strokes (lowest priority). The number of layers depends on how light or dark one needs the illustration to be.

The orientation of strokes is important when creating art because it helps to define an objects' shape and surfaces. In NPR, researchers have included direction fields in their work to enhance the results of their methods. Haeberli published one of the first articles in the NPR field to simulate a painterly style [25]. He developed a system that provides the user with different brush strokes considering the angle of the stroke to assign it a direction. Hertzmann [26] uses brush orientation to create paint-like images from photographs. The image gradient's normal are computed in order to place the strokes.

Later, he applied direction fields [27] to place hatching strokes on salient features of a smooth surface by computing its silhouette. Winkenbach et al. [65] extended their work to create this type of illustrations for curved surfaces. The main difference is the way the stroke textures are generated, e.g., the orientation and density of strokes is taken into account. In order to control the orientation of the strokes, they use isoparametric curves in a grid in the parametric domain (u, v).

Another approach for creating pen-and-ink illustrations interactively was proposed by Salisbury et al. [56]. The user selects among different stroke styles and defines the orientation of the strokes by drawing direction fields at every point of the grayscale input image which determines the tone to be achieved. The direction field described is similar to a pixel grid, where the direction at each position receives values between 0 and 255. The pattern of strokes has vertical orientation by default, therefore they need to be transformed, e.g., rotated, in order to follow the direction field. Figure 2.5 presents an overview of the process.

Kyprianidis et al. [31] have surveyed the 2D NPR algorithms that have been developed in the last twenty years. Many of the algorithms they have described are classified under the category of Stroke-Based Rendering and from their work we can see that most of the algorithms we have reviewed are part of this classification. One of the simplest methods to create illustrations is with line drawing and several methods have been developed for this purpose. Some of this methods are based on Image Filtering [31] by means of the Bilateral and Difference of Gaussians (DoG) filters.

Kang et al. [29] developed a technique to generate a line drawing based on photographs. This technique is a filter called Flow-based Difference-of-Gaussians (FDoG). They create a direction field called Edge Tangent Flow (ETF) based on the flow of important features of the images making the less important ones to follow this flow. The edge tangent is defined as a vector perpendicular to the input image gradient. Once the direction field has been computed, a Difference of Gaussians filter is then applied along the ETF. The output from this operation is used to generate a black and white image which yields a line drawing illustration. This method was later applied to create abstracted images from photographs [30] using a color and smoothing filter and ETF for line abstraction.

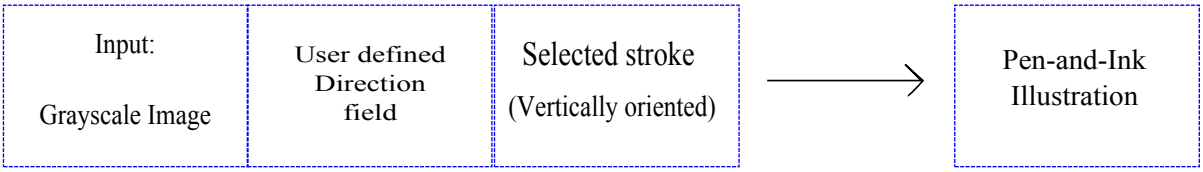


Figure 2.5: Overview of the system developed by Salisbury et al. [56] for generating Pen-and-Ink Illustrations

2.2.2 Simulation of Textures

One of the main objectives of NPR is the simulation of the different artistic styles such as illustrations applying diverse techniques, and painting. The evolution of NPR during the last ten years has allowed the development of new methods to simulate a greater variety of artistic styles [31]. Some of these new methods include region-based techniques based on the segmentation of the input image with rendering carried out according to the characteristics of each region. Other methods are example-based techniques which use a source image where textures and colors are transferred from. Lee et al. [34] have developed a method to render images with the painting technique known as texture transfer. Their approach takes texture from a reference image and places it according to a target image. The gradient of the target image is used to define the flow of the pixels and then apply the texture in this direction. The result is an image with the color of the target image and the texture extracted from the reference simulating brush strokes.

Non-photorealistic rendering of cloth has been little explored, this served O'Donovan and Mould [46] as motivation for developing an algorithm to create images with a felt-like look. They provide a description of the process to create felt-art in the real-world and take this information to create a method to simulate it. The fibers of wool are represented by 3D line segments which are placed in layers following a specific direction for a better definition of edges.

From the description of different NPR techniques above, we can see the importance of direction fields for the enhancement of the resulting images. Even though our goal is not the simulation of embroidery, one of the objectives of our research is to find a method to define the direction of stitches in order to obtain a high-quality embroidery design. Also, as mentioned, there exist some region-based methods, where the segmentation of the input is required, therefore, our research can be related to NPR in that sense.

2.2.3 NPR in Embroidery

Chen [12] has developed several techniques to generate images with hand embroidery-like look. This work was inspired by traditional embroidery techniques and motivated by the lack of algorithms to simulate this type of art in the field of NPR. One of the objectives was to produce embroidered images using different stitch styles that show the effects of light in real time as well as the generation of embroidery on 3D objects. The algorithm uses as input a raster scanned line drawing pattern just as the traditional real-world embroidery patterns. Then several methods are applied to obtain and label regions and outlines because they need to be rendered in different ways. A system was developed allowing the user to interact and change color and select among several embroidery styles. For this work, the author also needed to take into account orientation of the stitches, because different stitches may have different angles in order to produce an effect and also because of the way they look from different points of view.

The above could be considered the closest work to our research, however as mentioned in Section 2.2.2 we do not aim to simulate embroidery but analyze shape in order to segment it into regions and define the direction of the stitches. Our goal is to enable the use of these techniques in CAD systems for the creation of automated embroidery designs.

2.3 Shape Segmentation

In this section we provide a brief definition of shape segmentation which is an important technique in shape analysis and is widely used in different applications of computer graphics and computer vision. Because one of the goals of our research is to segment shapes into regions based on the shapes' skeleton we will discuss some literature describing methods that use this type of descriptors for this purpose.

2.3.1 Definition

Shape segmentation is defined as the decomposition of a shape into meaningful parts and is an important task in computer graphics and in computer vision applications such as object recognition, shape matching or shape classification. One of the goals of shape segmentation is to make these operations easier in comparison to the analysis based on the entire shape [15]. There are two classes of shape segmentation techniques: contour-based techniques and region-based techniques. Region-based techniques commonly use

shapes skeleton to carry out segmentation. The branches of the skeleton can be used to divide a shape into parts.

The human visual system analyzes and recognizes objects by decomposing them into parts. Some applications may need a deeper analysis of how shapes are described, for example how their parts are related to each other and how each part could be divided into other components [59]. The results from studies about how humans perceive objects in combination with the information obtained from the skeleton or from the boundary of the shape, or both, have been used as motivation for the development of different algorithms for shape segmentation. Mi and DeCarlo describe a method to remove parts from a shape [41]. Their approach is based on smooth local symmetries and on the concept of relatability which measures how easy two curve segments can be merged to create a smooth curve. They argue their method can be used in psychological studies related to visual association and shape transition.

The above could be considered as an example of the importance of taking into account the way humans identify objects. As mentioned before many algorithms for shape segmentation are based on this criterion in addition to the data provided by the shape descriptors. In the next subsection we will review some of these methods.

2.3.2 Skeleton-based Shape Segmentation

Zeng et al. [67] have proposed a method for shape segmentation based on the aforementioned, where they aim to decompose a shape simulating the human visual perception. The authors argue the branch points are important in the segmentation process because they allow the identification of protrusions and then classify them into "parts and non-parts". Once the branch points have been identified, it is necessary to find the points on the boundary that define line segments that decompose the shape. These points (tangent points) are found by using the maximal disk centered at the branch point. From the tangent points, the two points with higher negative curvature are selected. Another approach based on the human process for object analysis and recognition is proposed by Rom and Medioni [54] which is also a boundary-region-based approach where they combine different methods taking the most relevant properties of each one, using local symmetries of shapes for their representation, and segmenting them at curvature minima creating a hierarchy according to the size of the segments or parts.

As mentioned in Section 2.1.1, Shah describes a method called gray skeleton in [59]. In the same publication an algorithm for segmenting shapes based on this description is

also presented. The author first assumes that the branches of the skeleton are another way of representing shapes segments and then uses this concept to find the axes of ribbon-like components. The ribbons are defined as "strips of approximately constant width". A second approach is described where a shape is decomposed by finding the protrusions. The segmentation is carried out inside and outside the shape, as a result of the gray skeleton which is also internal and external. In Section 2.1.3 we also mentioned the work by Macrini et al. [35] where they describe a method to obtain a skeleton based on the concepts of ligatures and bones, this leads to the construction of a bone graph which is then used in the recognition of 3D objects.

Domakhina [15] devised an algorithm for segmenting a pair of shapes based on continuous skeletons and skeleton graphs. She refers to this process as isomorphic shape decomposition and argues that the segmentation of a shape could be directly related to the skeleton segmentation in a sense that each branch of the skeleton is mapped to a part of the shape. In this case the skeleton graph could be defined as a skeleton segmentation. It is also important to mention the following two definitions in order to understand the concept of isomorphic:

- " Two skeletons are isomorphic if their skeletal graphs are isomorphic and the traversal order of all vertices is the same in both graphs."
- "Two graphs are isomorphic if a mapping between their vertices keeps the edge adjacency."

Domakhina's method has as objective the comparison of two shapes to determine their similarity and can be used to solve morphing problems as well.

Reniers and Telea [53] proposed an algorithm to compute a curve skeleton including a method for segmenting 3D objects based on this skeleton. Geodesics are used to compute the skeleton and obtain a reliable segmentation. Junction points are also a key element for the segmentation of the objects, in this case, these points are called "critical points" which are determined by means of the curve skeleton or may be defined by the user. The proposed method also allows a hierarchical representation of the shape components which are numbered from 1 to n and classified into foreground and background parts, where n is the background and the remaining parts are considered as foreground. These parts are related to the critical points and are the most relevant ones.

Goh [21] presents several strategies that could help in tasks related to shape matching and one of the steps in the process described in this work is the segmentation of the skeleton. Goh first describes the Multiresolution Gradient Vector Field (MGVF) skeleton

which is defined by gradient vectors that "exhibit maximum directional disparity". One of the main characteristics of this skeleton is disconnection, where the most important branches are called the structural skeleton, while the branches originated by the noise of the boundary and are disconnected from the structural skeleton receive the name of textural skeleton. Once the skeleton has been computed, the next step is its segmentation, in order to do this, different principles are used. Two of such principles are the same than in other methods already described in this section, i.e., the branch points or skeleton junction identification and minima/maxima curvature. The other criterion used in this method is based on the gradient vector field where the minima/maxima directional disparity needs to be identified.

Most of the existing skeleton-based techniques for shape segmentation take into account the junctions of the skeleton branches as an important factor to decide where to segment a shape. Another criteria used for this purpose is the detection of negative curvature. From the above we can see that the skeletal representation of shapes provides us with useful information about the shape that helps us in the decision making process for its segmentation.

2.4 Summary

In this chapter we reviewed work related to our thesis on the topics of skeletal representations, Non-Photorealistic Rendering (NPR) and shape segmentation based on shape skeletons. In Section 2.1 we discussed the different techniques to compute a shape's skeleton, such techniques are classified into: grassfire simulation, thinning methods, skeleton based on Voronoi Diagrams and skeleton based on distance maps. In addition, we reviewed several algorithms that have been developed using these techniques. The skeleton is an important tool in computer graphics therefore its properties and some of its applications were also described.

Section 2.2 presented some work related to Non-Photorealistic Rendering, specifically we review some literature describing techniques that include the use of direction fields in their development such as the simulation of painting techniques and pen-and-ink illustrations. We also reviewed some algorithms for the simulation of textures. We discussed some work that aim the simulation of hand-made embroidery designs as well as the light effects on the designs. Our research is related to these concepts in a sense that we also developed a technique to give stitches a desirable orientation in order to obtain quality embroidery designs, however, our purpose is not the simulation of embroidery but

the application of different techniques in CAD systems used in the automatic embroidery process. Besides, as described in the literature review, some NPR algorithms are classified as region-based techniques, therefore, our research could be related to these kind of approaches.

Finally, we discussed different approaches for shape segmentation based on the skeleton. Most of these methods include the identification of junctions and boundary segments with negative curvature as criteria for the segmentation. This last section together with skeletonization methods provides us with important information that form the basis for our implementation.

Chapter 3

Skeletonization and Shape Analysis

The aim of our research is to find a method that allows the segmentation of 2D shapes based on its skeleton and its application in the creation of automated embroidery designs. In Section 3.1 we will describe the algorithm for the computation of a shape's skeleton that was used as basis for our implementation. The skeleton is calculated from an Integer Medial Axis which is a shape representation that maps between the boundary and the skeleton. This mapping will be discussed in Section 3.2. A summary is given in Section 3.3.

3.1 Shape Skeleton

Our work is based on the skeletal representation devised and implemented by Solis Montero and Lang [43]. The goal of their algorithm is to generate a pruned skeleton by means of contour approximation and the Integer Medial Axis(IMA). The main properties of the skeleton obtained from their approach are that it preserves the connectivity of the skeleton, respects the topology of the shape, is robust under the presence of noise and rigid shape transformations. It removes undesired branches and allows the reconstruction of the original shape. This algorithm is divided into four main steps: approximation of the shape's contour, computation of the skeleton, pruning and the vectorization of the skeleton. The computation of the skeleton in the second step is based on the Integer Medial Axis (IMA) by Hesselink et al. [28] which allows the computation of 2D images and 3D objects skeletons by means of the Feature Transform (FT). The FT follows the same principle than the Distance Transform: to assign each pixel on an image the distance to the closest boundary point. However the FT returns the closest point on the boundary

instead of the distance in case of the DT. The original algorithm by Hesselink et al. [28] for the IMA is also accompanied by three pruning methods: constant, linear and square-root pruning, in order to remove undesirable branches from the skeleton. Broadly speaking, once the FT has been computed, the algorithm for the IMA uses it to find the skeleton points. It loops over the 2D or 3D data and tests if the points are part of the boundary or not. If not, based on the FT, these points are tested to determine if they are part of the skeleton and if they need to be removed according to a pruning condition (e.g., constant, linear, square-root). One of the main advantages of this method is that it can be computed in linear time but the skeleton may be disconnected due to the pruning process. Next, we will describe the process for computing the Feature Transform and the IMA, and then we will give a step-by-step description of the skeletonization method in [43].

3.1.1 Feature Transform (FT)

The algorithm proposed by Hesselink et al. [28] uses the Euclidean distance to compute the FT. The input is a binary image defined on a rectangular grid **S**. There is a subset of S called the foreground **F** and the background **B** is the complement. Each point in the grid is represented by **p**. First, it is necessary to define the distance transform function *dt*, which finds for every point p the distance to the closest point on the boundary, in this case, the background. According to the authors *dt* is specified as follows

$$dt\,(p, B) = min\,\{\|p - y\|\,|y \in B\} \quad (3.1)$$

And define the FT function which assigns to every point p the set of closest points on the boundary

$$FT(p, B) = \{y \in B\,|\|p - y\| = dt(p, B)\} \quad (3.2)$$

They argue that the computation of the FT could be an expensive process and for this reason they limit their implementation to find for each point p only one feature transform point fp which is adequate for the computation of the IMA. To do this, the algorithm for the feature transform for 2D binary images is divided into two stages, one per dimension, and each stage consists of two inspections or scans.

The first part has as input a binary image of size $w \times h$ stored in a Boolean array where *true* is assigned to those points that belong to the boundary B. In part one, the first inspection is done row by row analyzing each point on the current row from right to

left, and an auxiliary array $dr[w]$ is used to store the distance to their closest boundary point. The second inspection follows the same process but this time from left to right, using $dr[w]$ to update the distance of each point to the boundary generating the function f_1 which is the 1D feature transform and the output of the first stage. Figure 3.1 shows the pseudo code for the first stage of the FT algorithm.

```
Input:  Let I[ ] be boolean array of a binary image of size w × h
Output: 1D feature transform f1
for each row h_i do
    for each column w_i from right to left do
        if  I[w_i, h_i] is boundary then
            d[w_i] = 0
        else
            d[w_i] = 1 + d[w_i + 1]
        end if
    end for
    for each column w_i from left to right do
        *using d[w_i] compute
        f1[w_i, h_i]
    end for
end for
```

Figure 3.1: Algorithm to compute the Feature Transform, first stage. Adapted from [28].

The second stage uses as input the f_1 computed in the first step in order to find the final feature transform f_2 in 2D and is done column by column inspecting each point first from top to bottom and then from bottom to top. A detailed explanation to compute the 2D distance transform is provided in [39] where the authors make a geometrical analysis for this second stage of the algorithm. They consider a mapping function M_w, $x \mapsto f(x, w)$ where the graph is a parabola for the Euclidean distance and the distance transform can be seen as the lower envelope of all the M_w's. The lower envelope is a set of curve segments, and each segment is expressed as $c[n]$ with n = 0, 1,...,k. Also, the projections of each segment on the x-axis generates a region, these regions or partitions are represented by $r[n]$. The separator function $Sep(u, v)$ is introduced to compute the intersection of functions M_u and M_v where $(u < v)$. All these functions are computed in the first scan of the second stage and then applied in the second scan for the computation of the f_2, see Figure 3.2, where $Sep(c[n], h_i) =$

$(h_i^2 - c[n]^2 + d(h_i)^2 - d(c[n])^2)/2 * (h_i - c[n])$ and d is the Euclidean distance, therefore $d(h_i)^2 = (w_i - f_1[w_i, h_i])^2$ and $d(c[n])^2 = (w_i - f_1[c[n], h_i])^2$.

```
Input:  feature transform f1
Output: 2D feature transform f2
for for each column w_i do
    n = 0, c[0] = 0, r[0] = 0
    *compute n, c[n] and r[n]
    for each row h_i from top to bottom do
        if n < 0 then
            n = 0, c[0] = h_i
        else
            val = 1 + Sep(c[n], h_i)
            if val < h_i then
                n = n + 1, c[n] = h_i, r[n] = val
            end if
        end if
    end for
    for each column h_i form bottom to top do
        *using f1, n, c[n] and r[n] compute
        f2[w_i, h_i]
    end for
end for
```

Figure 3.2: Algorithm to compute the Feature Transform, second stage. Adapted from [28].

3.1.2 Integer Medial Axis (IMA)

In this section we will describe the Integer Medial Axis by Hesselink et. al [28]. Its computation uses the feature transform ft to identify the skeleton points. Given an input image of size $w \times h$ and the feature transform, every point in the image is assigned one of three values defined as $WHITE$, $BLACK$ and $SKEL$ to differentiate the skeleton points from the foreground and background points. If a point p or its neighbor p_1 are part of the foreground (WHITE) they are tested in a method $compare(p, p_1)$ to determine if they are part of the skeleton. In this method, the feature transform of p and p_1 is found: $f_p = ft[p]$ and $f_{p1} = ft[p1]$. If the distance $\|f_p - f_{p1}\| > 1$ and if a pruning condition (depending on the pruning method selected) is satisfied, the points are tested to verify if

at least one of them has two equidistant points on the boundary. This property holds for points on the skeleton in the IMA. As described in [28] a point p is part of the skeleton if $\|f_{p+x} - f_p\| > 1$ and $\|m - f_{p+x}\| \leq \|m - f_p\|$ where $m = p + \frac{1}{2}x$ and is defined as the middle point of the segment p, p+x (see Figure 3.3).

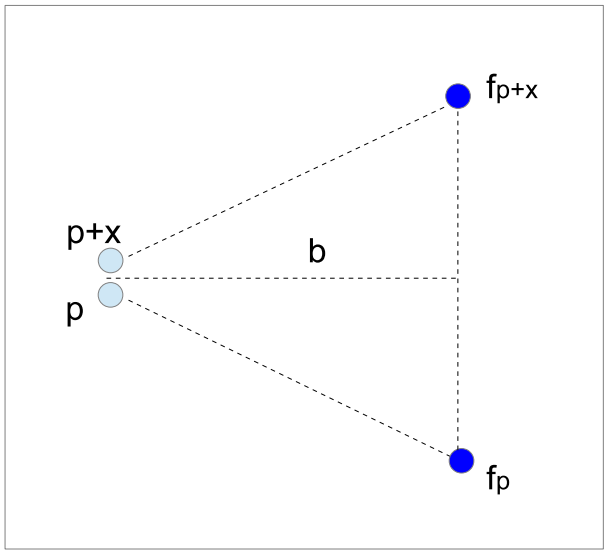


Figure 3.3: The closest point to the perpendicular bisector b of the segment $f_p f_{p+x}$ is defined as a skeleton point. If p and $p + x$ are at the same distance to the bisector, then both of them are considered as part of the skeleton.

3.1.3 Contour Approximation

The input of the algorithm is a binary image which is processed in this first step and approximated by piecewise curves. According to Solis Montero and Lang [43], when it comes to contour approximation one of the problems is to specify a suitable number of knots in order to obtain a reliable approximation. In their algorithm, they do not need to determine the number of knots to be placed and use the "split-and-merge" technique devised by Ramer [52] for the knot placement and curve approximation. Two types of approximations are discussed in [43]: linear, which uses line segments to approximates a set of points, and cubic, which makes use of cubic Bezier curves. In this step a threshold T is required to measure the error allowed between the approximation and the original shape.

First, it is necessary to extract a subset of pixels from a given shape contour "$C = c_1, c_2, ..., c_n$" and consider the points in the subset now as the end points of the curve

segments. The first two points or knots selected and approximated by a curve are the first and last pixels of the input contour, that is c_1 and c_n. Next, the minimum distance d_{min} from each pixel to the curve is computed. The pixel c_k with the maximum d_{min} is identified. If its distance to the curve is greater than the threshold T, then the curve segment is divided at c_k into two segments. This is done until the distance of a pixel c_k to the approximated curve is less than T. The output of this process is a set of knots and an approximated shape with error of T. The above is illustrated in Figure 3.4.

The curve approximation uses parametric Bezier curves and is done by means of the least-square-fitting technique. In this part of the algorithm, two consecutive knots are taken into account to represent the end points of the Bezier curve. In order to compute the middle control points, the original contour points between these two knots are used for the curve fitting. The least-square fitting is given by the following

$$S = \sum_{i=1}^{n} [p_i - q(t_i)]^2 \tag{3.3}$$

Where n is the total number of input points p and $q(t_i)$ is the parametric Bezier curve, then 3.3 can be expressed

$$\begin{aligned} S = \sum_{i=1}^{n} (p_i - (1-t_i)^3 P_0 + 3t_i(1-t_i)^2 P_1 \\ +3t_i(i-t_i)P_2 + t_i^3 P_3)^2 \end{aligned} \tag{3.4}$$

P_0 and P_3 are the ends control points, while P_1 and P_2 are the middle control points and can be found by considering $\delta S/\delta P_1 = 0$ and $\delta S/\delta P_2 = 0$ then,

$$\begin{aligned} P_1 &= (A_2C_1 - A_1C_2)/(A_1A_2 - A_{12}A_{12}) \; and \\ P_2 &= (A_1C_2 - A_{12}C_1)/(A_1A_2 - A_{12}A_{12}). \end{aligned}$$

A_1, A_2 and A_{12} are defined by

$$
\begin{aligned}
A_1 &= 9\sum_{i=1}^{n} t_i^2(1-t_i)^4, \\
A_2 &= 9\sum_{i=1}^{n} t_i^4(1-t_i)^2 \ \textit{and} \\
A_{12} &= 9\sum_{i=1}^{n} t_i^3(1-t_i)^3.
\end{aligned}
$$

Finally,

$$
\begin{aligned}
C_1 &= \sum_{i=1}^{n} 3t_i(1-t_i)^2[p_i-(1-t_i)^3P_0-t_i^3P_3] \ \textit{and} \\
C_2 &= \sum_{i=1}^{n} 3t_i^2(1-t_i)[p_i-(1-t_i)^3P_0-t_i^3P_3].
\end{aligned}
$$

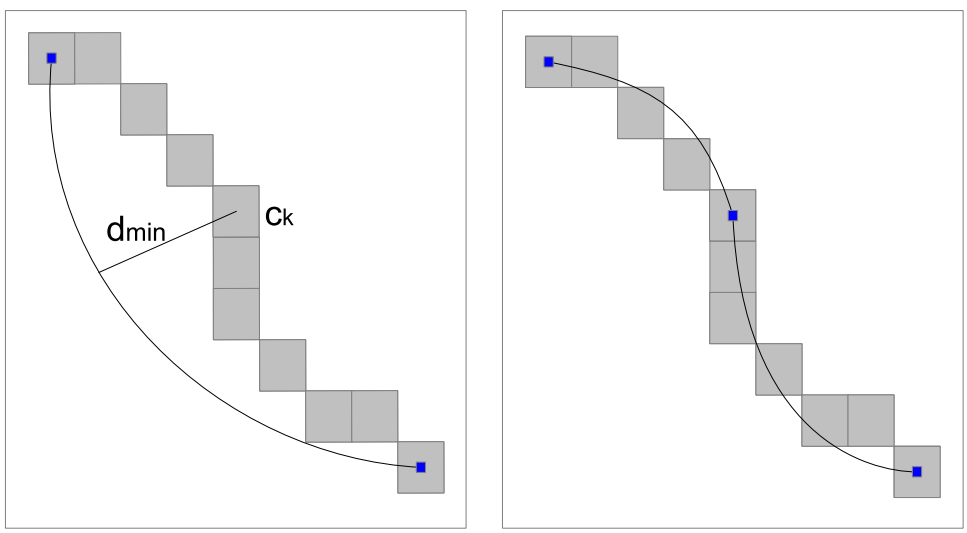


Figure 3.4: Contour Approximation. The curve is divided into two segments at c_k, the pixel with maximum d_{min}.

3.1.4 Skeleton Computation

During the contour approximation each curve segment is assigned an ID which is used in the second step to find the skeleton points. In this second step the skeleton is computed by means of the Integer Medial Axis. As described in the last section, the IMA can

be pruned by selecting among three different pruning methods. In the method we are describing another pruning method is introduced. The idea is that for a point to be part of the skeleton it needs to have two equidistant points on two different boundary curves which are on two different sides of the shape. As described in [43], this pruning condition removes undesirable branches leaving only those ones with an end point at an intersection of two boundary curves. These branches are treated for removal in the third step of the algorithm which is described in the next section.

Same than in the original IMA, given two points x and its neighbor y if at least one of them belongs to the image foreground, their corresponding feature transform fx and fy is identified. Then, they are tested to verify if they are on different boundary curves using the curve ID in order to decide if x and/or y are part of the skeleton. See Figures 3.5 and 3.6.

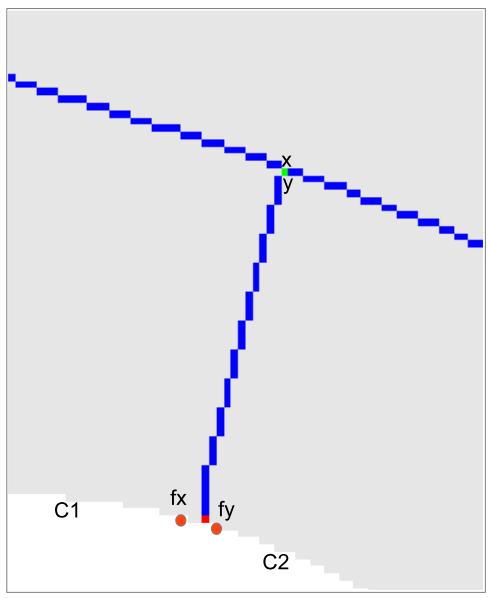


Figure 3.5: Selection of Skeleton points. A point x in the image is selected if there is a neighboring point y with feature transform fy on a different boundary curve than fx.

3.1.5 Skeleton Pruning

It was mentioned in the last section that after the skeleton is computed some minor branches due to small shape boundary perturbations remain and therefore a pruning step is required in order to remove them. Two types of points need first to be identified: branch points and end points. This classification is done according to the number of

neighboring skeleton pixels they have. Branch points have three or more neighboring pixels while end points have only one neighbor. The branches that need to be removed from the skeleton are defined by a branch point *bp* and an end point *ep* if the skeleton between *bp* and *ep* does not include any other branch point. Consider the feature transform ft of the *bp* and a circle centered at *bp* with radius $r = |ft - bp|_2$, if *ep* lies inside of this circle, the branch is considered minor and marked to be removed. Another threshold is introduced in this step, consider the threshold s as a scale factor of the radius r, then a branch will be removed if it is contained inside the scaled circle. After marking the branches for removal, the points that form the branch bp,...,ep are removed excluding bp. The above is done until no more branches can be removed giving as output a skeleton with no unwanted branches (see Figure 3.7).

$$|ep - bp|_2 \leq s * r \tag{3.5}$$

Input: image point x and its neighbor y
$compare(x, y)$
$fx = ft[x]$, $fx = ft[y]$
$C1 = C[fx]$, $C2 = C[fy]$
if $\|fx - fy\|^2 > 1$ AND $C1 \neq C2$ **then**
  $condition = inprod(fx - fy, fx - fy - x - y)$
  if $condition > 0$ **then**
    $skeleton[x] = SK$
  end if
  if $condition < 0$ **then**
    $skeleton[y] = SK$
  end if
end if

Figure 3.6: Procedure *compare* that specifies the conditions to identify the skeleton points. C1 and C2 are the curves ID. Adapted from [43].

3.1.6 Vector Representation of the Skeleton

The skeleton produced in Section 3.1.5 is formed by branches that can be defined by one branch point and one end point or by two branch points and in some cases where the input is a small shape, the skeleton is a single branch defined by two end points. Each of this branches is approximated by cubic Bezier curves following also the least-square

fitting technique described in the contour approximation. In this last step the Euclidean distance d from each skeleton point p to the approximated curve is computed. The maximum distance $dmax$ is identified, that is the distance from the pixel with maximum distance to the approximated curve. If $dmax$ is greater than the distance from a point p to its feature transform point f_p and if it is greater than the threshold T, then the curve is divided into two at point p.

$$(\|f_p - p\| \ \ and \ T) < d_{max} \tag{3.6}$$

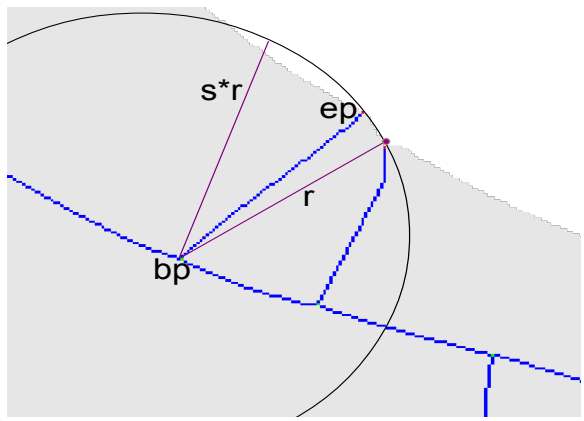


Figure 3.7: Branch removal according to Equation 3.5.

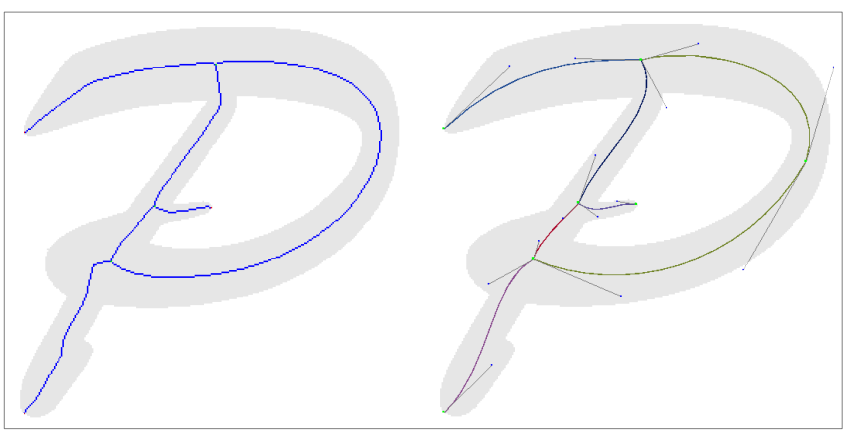


Figure 3.8: Skeletal Representation Example. $Left$, Final skeleton. $Right$, Vectorized Skeleton.

3.2 Skeleton Analysis

In Chapter 2 we mentioned it is necessary to do a shape analysis for our research and that it can be done by means of the skeleton which is a simplified way to represent a shape. One of the desirable characteristics for a shape skeleton is that it enables the reconstruction of the original shape from the skeleton. This is the case of the method described in Section 3.1, where the skeleton is computed based on the Feature Transform and provides us with the necessary information about the shape therefore it is possible to do not only a region-based but a boundary-based analysis. As part of this analysis we performed a boundary-skeleton mapping in the discrete and continuous domain, to determine the way every skeleton point is related to the boundary, not just by the Feature Transform but by the curves that approximate the shape. Next we will describe this mapping.

3.2.1 Point-on-the-Curve Algorithm

The algorithm to compute the closest point on a curve described in [20] represents a very useful tool for our implementation as we will see in the next sections. The main idea is to find the closest point to a query point, on a given curve. The closest point is defined by the value of the parameter $t_{min} \in [0, 1]$. The author uses the geometric approach of Figure 3.9 where the line segment $QC(t)$ is perpendicular to the tangent of the curve at $C(t)$ and can be expressed

$$[C(t) - Q] \cdot C'(t) = 0 \tag{3.7}$$

Where $C(t)$ is of degree three and $C'(t)$ is of second degree therefore, 3.7 results in a polynomial of fifth degree. This is, given a Bezier curve and an arbitrary point Q, the polynomial is converted into a fifth degree expression, then the roots are found by comparing the distances from Q to each solution and to $t = 0$ and $t = 1$. Once the minimum distance has been found, the algorithm returns the value of t_{min} and the closest point on the curve at that value.

The curve $C(t)$ can be expressed by means of the Bernstein-Bezier polynomials

$$C(t) = \sum_{i=0}^{n} P_i B_i^n(t) \quad t \in [0, 1] \tag{3.8}$$

where B_i are the polynomials of degree n

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad i = 0, ..., n \tag{3.9}$$

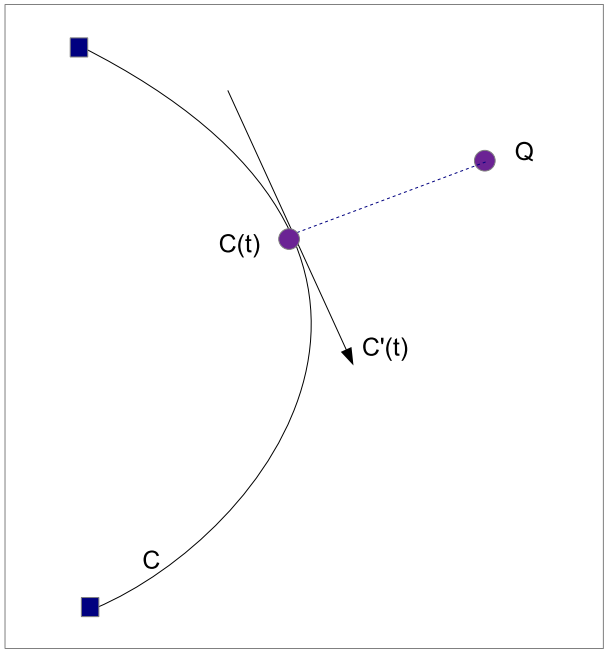


Figure 3.9: Geometric approach to solve the problem Point-on-the-Curve. Adapted from [20].

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{3.10}$$

and P_i are the Bezier curve control points. $C'(t)$ can be expressed in the same way than $C(t)$:

$$C'(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) B_i^{n-1}(t) \tag{3.11}$$

Substituting 3.8 in 3.7 we have

$$C_1(t) = \sum_{i=0}^{n} P_i B_i^n(t) - Q \tag{3.12}$$

Then,

$$\left(\sum_{i=0}^{n} P_i B_i^n(t) - Q\right) \cdot \left(n \sum_{j=0}^{n-1} (P_{j+1} - P_j) B_j^{n-1}(t)\right) = 0 \tag{3.13}$$

Finally the fifth degree polynomial can be defined

$$\sum_{i=0}^{n} \sum_{j=0}^{n-1} w_{i,j} B_{i+j}^{2n-1}(t) = 0 \tag{3.14}$$

where $w_{i,j} = c_i \cdot d_j z_{i,j}$, $z_{i,j} = \binom{n}{i}\binom{n-1}{j} \Big/ \binom{2n-1}{i+j}$, $c_i = P_i - Q$ and $d_i = n(P_{i+1} - P_i)$. $w_{i,j}$ is one of the components of the control points (y), then the other component (x) is $i + j/n + (n-1)$.

The algorithm for finding the roots of a high-order equation is described next: Once the equation has been converted into a Bernstein-Bezier form, the roots are found by computing the intersection of the Bezier curve with the horizontal axis. The x and y components of the curve are given by

$$\begin{aligned} X &= \sum_{i=0}^{n} x_i B_i^n(t) \\ Y &= \sum_{i=0}^{n} y_i B_i^n(t) \end{aligned} \tag{3.15}$$

For a parametric polynomial we have $B(t) = (x(t), y(t))$ then, the roots are the values of t where the curve intersects the $t - axis$. In order to find them, several properties of the Bezier curve are taken into account [19]:

> "The curve itself will always lie within the convex hull of the control points.
> The curve is an approximation to the control polygon that is "smoother" than the control polygon.
> Repeated subdivision of the curve yields control points that converge on the curve itself in the limit."

The recursive algorithm for the root computation is shown in Figure 3.10. The number of sign changes is translated as the number of times the control polygon crosses the t-axis. If there are no changes in the sign it means the curve does not cross the horizontal axis. The condition "if the control polygon is flat enough" refers to the scenario where the control polygon is close to a straight line. The algorithm can be stopped when this condition is satisfied. In order to determine the flatness of a control polygon, assume

```
Roots(P)
P = array[0,..,n] of control points
if (cp does not cross t-axis) then
    //cp = control polygon
    return null
else if (cp crosses t-axis) and (cp sufficiently flat or no more recursion allowed) then
    return t-intercepted from P[0] to P[n] as the root
else
    divide Bezier curve at midpoint by deCasteljau's algorithm
    Roots(left half)
    Roots(right half)
    return the solutions generated
end if
```

Figure 3.10: Root-finding algorithm. Adapted from [19].

a line connecting the first and last control points and a bounding box is defined where its top and bottom are parallel to the line, then the error accepted for the solution is determined by half the distance between the intersection points of the bounding box with the t-axis. If the error is acceptable, the intersection of the line with the t-axis is computed and returned as the root.

3.2.2 Boundary-Skeleton Mapping

The feature transform computed for the shape skeletal representation we are using leads to a mapping where every point in the shape is related to its closest point on the boundary and from the description of the skeleton computation we know that the skeleton points have two or more closest points on the boundary. In order to have a complete connection skeleton-boundary we carried out a mapping from the boundary to the skeleton.

It is necessary to recall that the skeleton is formed by several branches and each branch can be defined as a set of pixels, then when the skeleton is vectorized and approximated by cubic parametric Bezier curves, each curve is associated with a pixel chain. Hence, each pixel $p(x, y)$ is mapped to a point represented by the parameter t on a curve $S(t)$. The algorithm described in Section 3.2.1 is used to compute the point on the curve.

The boundary is processed in the same way than the skeleton, we have a set of contour points and from the contour approximation there is a set of contour curves which are

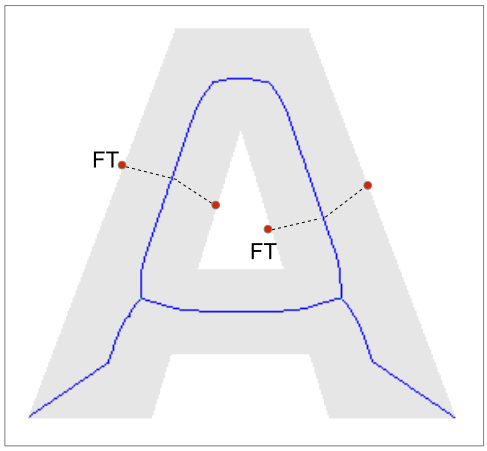


Figure 3.11: Each point on a shape's skeleton maps to its closest points on the boundary (FT points).

also cubic parametric Bezier curves. With these data sets it is possible to find for each point $c(x, y)$ on the contour its nearest point on a boundary curve $B(q)$ and the value of parameter q at that point.

With the help of the feature transform we can find each boundary curve segment and each point on the curve that maps to each point on the skeleton. This data is then organized by boundary curve: for each curve we know which boundary points and which skeleton points and curves are associated to it. As result we are able to determine for every value of q its related t value.

In our work we use this information to implement a mouse-based user interaction tool. When the mouse is clicked and dragged on the screen, we can see the point on the boundary curve (cyan circle), the point on the skeleton curve (black circle) and the point on the pixel chain (middle red circle) with the closest feature transform points at both sides of the skeleton (at inner and outer contour in some cases). All these points are the closest ones to the current position of the mouse (see Figure 3.12).

Another feature of this implementation is the possibility of obtaining the relation between a boundary and a skeleton curve. By double clicking on a boundary curve, a window with a graph describing such a relation is opened. For example see Figure 3.13 where the different values of parameter t are represented on the horizontal axis and their corresponding q values on the vertical axis. There are cases where several skeleton curves are mapped to the same boundary curve.

3.3 Summary

In this chapter we presented the algorithm to obtain a shape's skeleton which is the basis for our research. The technique described here is divided into four steps: contour approximation, skeleton computation, skeleton pruning and skeleton vectorization. We also presented a mapping between the boundary and the skeleton where we analyzed the information provided by the feature transform and how it can help us to understand the way the boundary is related to the skeleton. As part of this analysis, we implemented an user interface that allows us to see this relation by means of graphs. Some auxiliary algorithms, such as finding the nearest point on a Bezier curve and computing the root of a polynomial equation were also included.

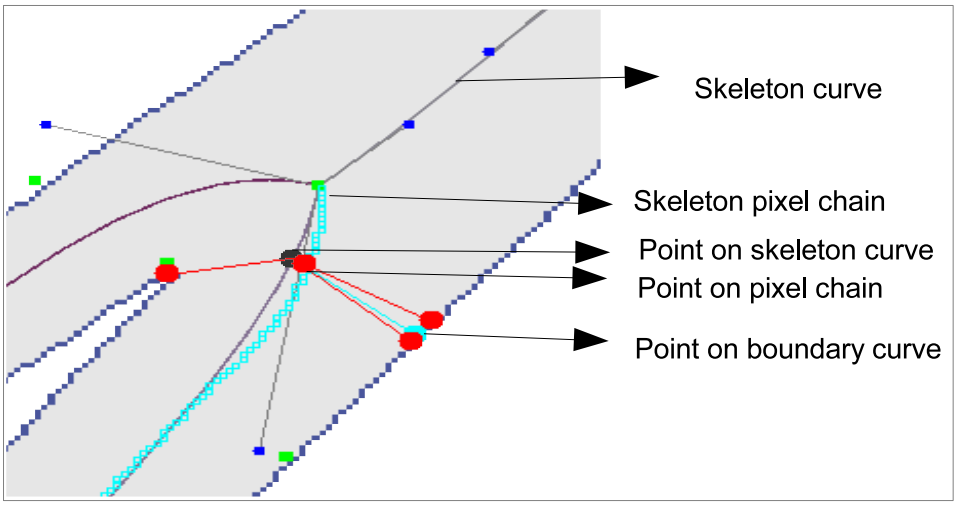


Figure 3.12: Example of mouse interaction. Closest point on the skeleton curve and closest point on the boundary curve.

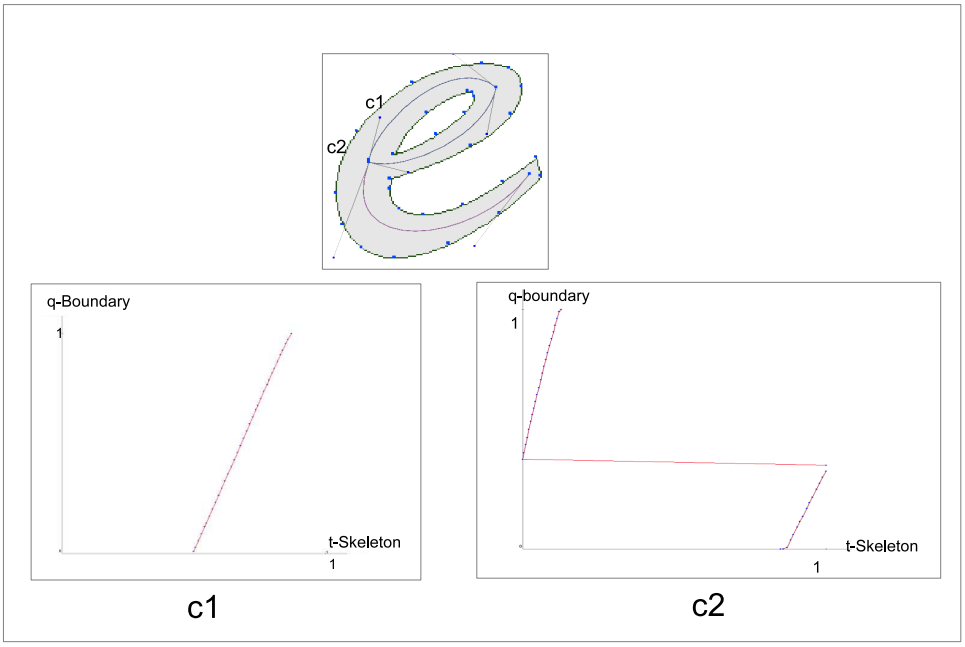


Figure 3.13: Relation between the skeleton and the boundary curves. In the graph on the left, only part of a skeleton curve maps to a boundary curve while in the graph on the right, several skeleton curves map to the same boundary curve.

Chapter 4

Shape Segmentation

In this chapter we will describe the process to segment shapes into several regions using the information obtained from the skeleton and the feature transform discussed in the last chapter. In Section 4.1 we present the concept of using lines as segment boundaries. The criteria we used for the placement of the segment boundaries is described in Section 4.2. There are some special cases where the computation of the lines needs an extra processing step which is discussed in Section 4.3. For our purpose, we need to compute the intersections of the segment boundaries with the shape boundary curves. This step is described in Section 4.4. In Section 4.5 we compare our segmentation technique with other methods from the literature. Finally, a summary is presented in Section 4.6.

4.1 Linear Segment Boundaries

Once the skeleton of the input shape has been obtained we need to develop a method for segmenting the shape into parts or regions. Several techniques were reviewed in Chapter 2. In some of these techniques the human visual perception has been taken into account to perform a reliable segmentation. Also, the information from the boundary and the skeleton is of paramount importance. This includes mainly the identification of branch points and protrusion [67] to select some points on the boundary for the placement of line segments to divide the shape. The identification of such points is done by means of the maximal disk centered at a branch point and by computing the curvature minima. Our method is similar than the one mentioned above in that we also need to find points on the boundary to define line segments that allow us to divide the shapes into several regions. These linear segment boundaries (SL) and the steps to find them will be described in

the next section.

4.2 Placement of Segment Boundaries

The process for the placement of segment boundaries is based on the vectorized skeleton obtained by means of the method described in Chapter 3. This shape representation is formed by a set of curve segments defined by either a branch point and an end point or by two branch points and in few cases by two endpoints (the skeleton is a single curve segment), also each segment is related to a pixel chain. Hence, the first part of the algorithm identifies the branch points and the curves emanating from each of them. Then, the curves that are more continuous at each branch point are "merged". The merging decision is made by using the tangent lines of the Bezier curves and computing the angle between them. If the angle is minimum, the curves are merged (Figure 4.1). The process is repeated until no more curves can be merged [1]. This criteria is then used for our shape segmentation method.

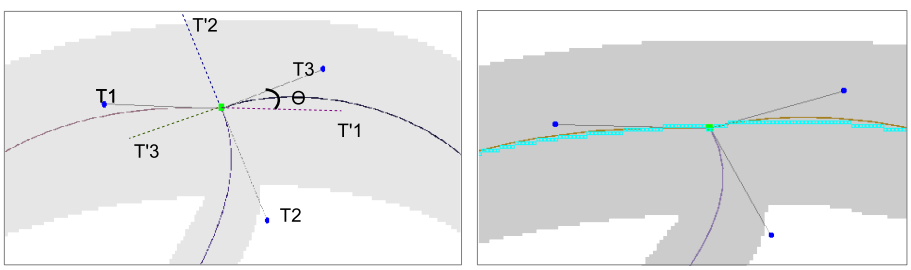


Figure 4.1: The minimum angle θ between the curves' tangent line segments T1,T2 and T3 is computed. The curves whose lines present this minimum angle are merged. The merge result is shown on the right, where the pixel chains (cyan) of the two merged curves appear like a single chain.

Once we have identified the branch points and the curves related to them, we need to know which of these curves remained unmerged because they will be used for the placement of the segment boundary lines. Then, for each branch point, we obtain its eight neighboring pixels and their feature transform points. By doing this, we know the closest boundary points around the branch point (at different 'sides' of the shape, see Figure 4.2). Figure 4.3 shows the pseudocode for this step. The input image is stored

[1] The merging algorithm is based on a preliminary implementation by Solis Montero and Lang.

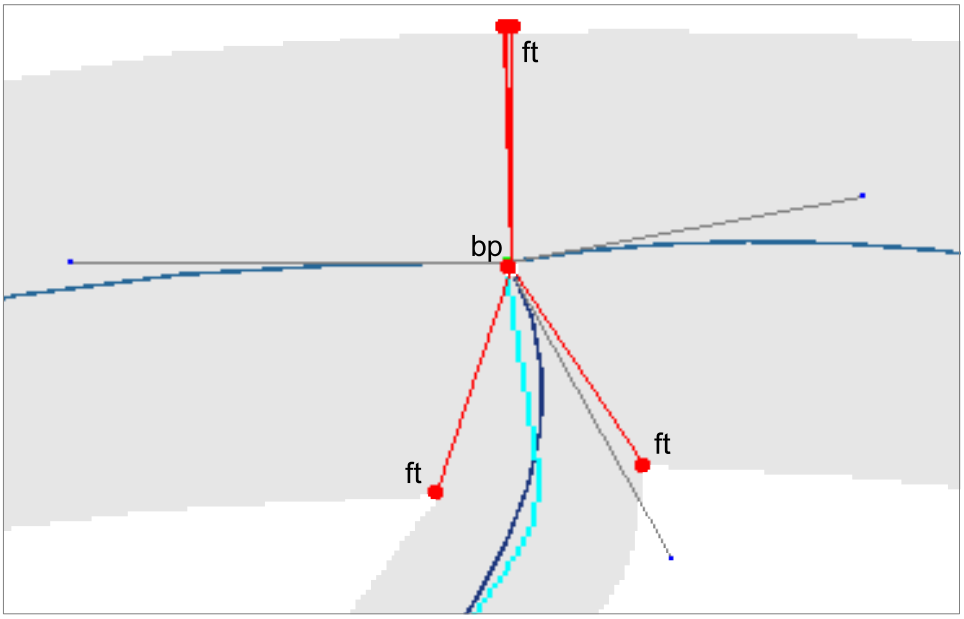


Figure 4.2: Branch point and the feature transform of its neighboring pixels.

Input: ft_2, branch points set, skeleton curves
Output: List of segment boundary lines (SL) and intersections (IS) with the boundary
for each branch point bp **do**
  C =unmerged curve
  $bpIndex$ =bp position
  $Points[\,]$ =bp's 8 neighors' ft_2 points
  $SL = getSL(bp, Points[\,], c)$
  $Lines.add(SL)$
end for
$IS = computeIntersections(\,)$

Figure 4.3: Algorithm to compute Segmenting Lines.

in an array thus, we need to compute the position of the branch point in the array. The method $getSL$ is described next.

4.2.1 Computation of Segment Boundaries Lines

In this section we will describe the method $getSL(\ )$ which takes as parameters the current branch point bp, the feature transform of its eight neighboring pixels, stored in $Points[\ ]$ and the unmerged curve c. The feature transform of some of these points may be the same so we only store those feature transform points that are different. The objective of this part of the algorithm is to find two points among the ones in $Points[\ ]$ to define a segment boundary line SL.

First, we obtain the tangent line segment of c that is defined by $P1 = bp$ and $P2 = CtrlP$, the control point of c closest to bp (see Figure 4.4). The tangent line segment equation in its general form $Ax+By+C = 0$ is computed and evaluated with each of the feature transform points ft (candidates) in $Points[\ ]$. We compute the distance d from each of these points to $P2$ and the dot product between the *tangent line segment* and the line segment defined by bp and each ft point. These values are needed to find two points in the same direction than the tangent line segment but each of them on different side of the line, and are stored in a way that all of them are related by an index in order to make the identification of the two required points easier.

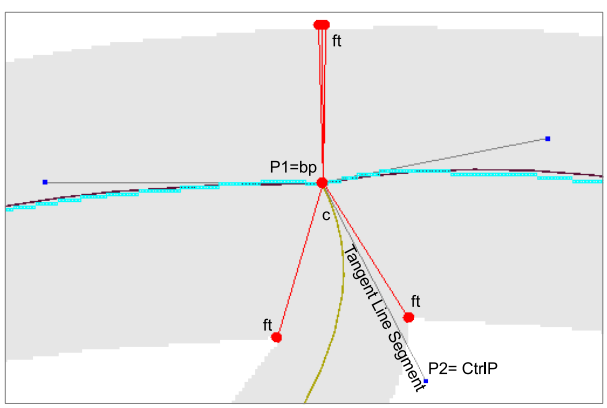


Figure 4.4: Identification of bp, feature transform points, unmerged curve c and *tangent line segment*.

When we have, for example, three candidates, it is easy to identify the two points we need by means of the dot product, that is by selecting the two largest values. If we have more than three candidates, we first select those whose normalized dot product is greater than zero, this means the lines that are pointing in the 'same direction' than the tangent line segment. Then, if the number of lines is equal to two, we have found the two points we are looking for. Otherwise if the number of lines is greater than two, we classify the values obtained from the line equation evaluation into positive and negative. For the positive values we find the index whose related line segment $\overline{bp, ft}$ contains the closest ft point to $P2 = CtrlP$. The same is done for the negative values. In this way we have identified the two ft points we are looking for to define a segment boundary line SL. This process is done for each branch point in the input shape. Figure 4.5 shows an example of a shape with its segment boundary lines (in red) and Figures 4.6 and 4.7 contains the pseudocode for this method.

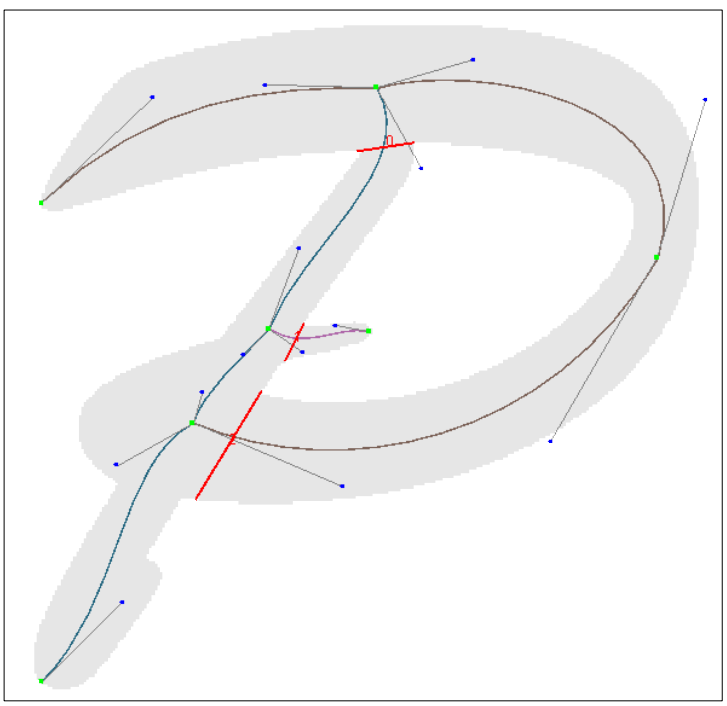


Figure 4.5: Shape and Segment Boundaries (red).

```
input: bp, c, Points[ ] = feature transform points of the 8 neighbors of bp
output: Segment boundary line SL
if c.P1 = bp then
    Tline = (bp, c.CtrlP1) //tangent line segment definition (TL)
else
    TLine = (bp, c.CtrlP2)
end if
lineEq[ ] = Tline.getEquation( ) //line equation in its general form
for each ft in Points[ ] do
    value = lineEq.evaluate(ft) //Evaluation of the line equation with the ft point
    d = distance(Tline.P2, ft) //Euclidean distance from CtrlP to ft
    l = line(bp, ft)
    dot = dotProduct(Tline, l) //normalized dot product
    dots.add(dot)
    values.add(value)
    dist.add(d)
end for
```

Figure 4.6: Pseudocode for *getSL*( ) method, part (a).

```
if Points[ ] <= 3 then
    //Find the two largest dot product values
    index1 = largest1
    index2 = largest2
else
    for i = 0 to i < dots.size do
        //Find index of fts in the same direction form bp than TL
        if dots(i) > 0 then
            index.add(i)
        end if
    end for
    for j = 0 to index.size do
        id = index(j)
        d1, d2 = threshold
        if values(id) > 0 then
            //Find the closest ft point to TL on its positive side
            valP.add(values(id))
            if dist(id) < d1 then
                d1 = dist(id)
                index1 = id
            end if
        else
            //Find the closest ft point to TL on its negative side
            valN.add(values(id))
            if dist(id) < d2 then
                d2 = dist(id)
                index2 = id
            end if
        end if
    end for
end if
SL = (ft(index1), ft(index2))
slicedCurve.add(c, LineCurveIntersection(c, SL))
```

Figure 4.7: Pseudocode for *getSL*( ) method, part (b).

The method *LineCurveIntersection*( ) computes the intersection of the segment boundary line with the skeleton curve therefore, *slicedCurve* stores the skeleton curve with the value of the parameter t at the intersection point. This method is described in the next section. The method *computeIntersections*( ) in Figure 4.3 finds the intersec-

tion points of the segment boundary line with the shape boundary and will be explained in Section 4.3.

4.2.2 Line-Cubic Bezier Curve Intersection

Given a cubic Bezier curve c and a line sl, the intersection of the line with the curve is computed, as well as, the value of the parameter t at the intersection point. First, we need to obtain the line equation in its general form $ax + by + c = 0$ and the polynomial form of the curve $B(t) = At^3 + Bt^2 + Ct + D$. Because we are dealing with cubic Bezier curves (Figure 4.8)

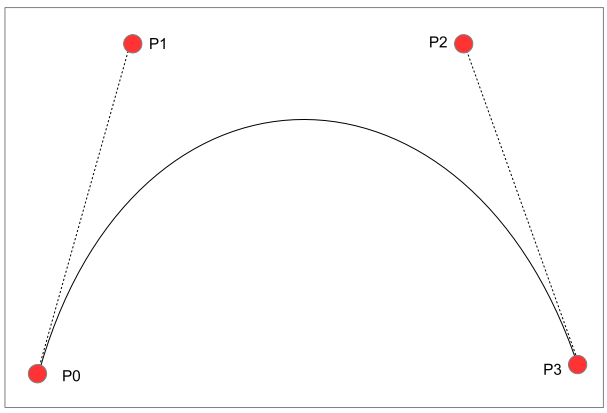


Figure 4.8: Example of a cubic Bezier curve.

the coefficients are given by

$$\begin{aligned} A &= (P_3 - P_0) + 3(P_1 - P2), \\ B &= 3(P_2 - P_1) - 3(P_1 - P_0), \\ C &= 3(P_1 - P_0) \ and \\ D &= P_0. \end{aligned} \tag{4.1}$$

After computing the coefficients for the curve equation, we replace x and y in the line equation for the X and Y components of the curve, resulting in a third degree equation (lcE) in t. There will be three roots for the equation but we are only interested in a solution between 0 and 1 to remain on the curve. If such root exists, we evaluate the curve equation with its value in order to obtain the intersection point. The value of the parameter t at the intersection is given by the root. To find the roots of a cubic equation is not as simple as finding the root for a quadratic equation therefore, we made use of the algorithm by Jochen Schwarze [19]: Given a cubic equation

$$a_3x^3 + a_2x^2 + a_1x + a_0 = 0 \tag{4.2}$$

Every term is divided by a_3 resulting in an equation of the form

$$x^3 + Ax^2 + Bx + C = 0 \tag{4.3}$$

$$x = y - A/3 \tag{4.4}$$

If x in 4.3 is substituted by 4.4 we obtain

$$y^3 + 3py + 2q = 0 \tag{4.5}$$

As we can see the quadratic term has been eliminated and the determinant is given by

$$D = q^2 + p^3 \tag{4.6}$$

$$u, v = \sqrt[3]{-q \pm \sqrt{D}} \tag{4.7}$$

and the roots are defined by

$$\begin{aligned} y_1 &= u + v \\ y_{2,3} &= -\frac{u+v}{2} \pm \frac{\sqrt{3}}{2}(u-v)i \end{aligned} \tag{4.8}$$

If $D > 0$, one of the root is real and the other two are conjugated complex numbers. If $D = 0$ only two real values are obtained as roots and if $D < 0$ three different roots that are real numbers are obtained, Schwarze uses a trigonometric approach to compute three real roots by

$$\begin{aligned} cos\theta &= -\frac{q}{\sqrt{-p_3}}, \\ y_1 &= 2\sqrt{-p}\ cos\frac{\theta}{3} and \\ y_{2,3} &= -2\sqrt{-p}\ cos\frac{\theta \pm \pi}{3}. \end{aligned} \tag{4.9}$$

As final step, a resubstitution of $(1/3)A$ is used to obtain the right values of x (roots). Sometimes we obtain more than one point whose value of the parameter t is within the interval $[0, 1]$, so we need to select the most convenient one. In order to do that we

compute the middle point (mp) of the segment boundary and the distance from it to each solution point, then we compare them to find the closest point to the middle. The skeleton is centered in the shape therefore, the point on the line that intersects the skeleton curve should be closed to middle of the line. For this reason we picked the midpoint as reference to select the intersection point. The algorithm is shown in Figure 4.9.

4.3 Special Cases

From the description of the process to obtain the segment boundaries we can see the importance of the skeleton as we have pointed out in the last chapter. In most cases our implementation presents good results however, there are some scenarios where the skeleton configuration does not allow a proper shape segmentation. This section describes two cases where the results obtained are not favorable.

4.3.1 Extra Pruning

In Chapter 3 the algorithm to compute the skeleton was described. While the skeleton points are found, some unwanted branches are removed leaving those that have their end points at one of the points where two approximated boundary curves join, hence a pruning step is required after the skeleton computation. This pruning only considers for removal branches defined by a branch point bp and an end point ep, under the condition that they do not contain any other bp.

In some shapes there exist squared ends where the pruned IMA has branches ending at each corner. For our purposes this is not always desirable therefore, we implemented an extra skeleton pruning. To illustrate the above, consider Figure 4.10 where we present two shape examples, their skeleton and the resulting segmentation boundaries before adding the pruning step.

According to the segmentation process, the curves at a branch point that present more continuity were joined and left a branch unmerged. This branch was used to place a segmenting line but in this case the segmentation is not as desired. We improve the segmentation by removing branches before the actual unmodified segmentation step as previously described. From Figure 4.10 it can be seen that this additional processing is needed for the shortest branches between a branch point and an endpoint. Therefore, the first part of the pruning step is to identify the end points and their corresponding

```
Input: Skeleton curve c and segment boundary line sl
Output: nearest = intersection point and t value
sol[ ] = 0
lineEq = LineEquation(sl) //segment boundary line equation
cX = curveX(c) // curve polynomials for X and Y components
cY = curveY(c)
lcE = Line_Curve(cX, cY, lineEq) // cX and cY substitute the line x and y
//Solve 3rd degree equation and store the solutions in sol[ ]
n = SolveCubic(lcE, sol[ ])
for i = 0 to i < n do
    //Look for the solutions between 0 and 1
    if sol[i] ≥ 0 AND sol[i] ≤ 1 then
        x = (cX[0] * sol[i]^3) + (cX[1] * sol[i]^2) + (cX[2] * sol[i]) + cX[3])
        y = (cY[0] * sol[i]^3) + (cY[1] * sol[i]^2) + (cY[2] * sol[i]) + cY[3])
        result[i] = (Point(x, y), sol[i])
    end if
end for
mp = sl middle point //compute sl middle point
d1 = threshold
for j = 0 to j < result.length do
    dist = distance(result[j].point, mp)
    if dist < d1 then
        nearest = result[j]
        d1 = dist
    end if
end for
return nearest
```

Figure 4.9: Algorithm to compute the intersection of the segment boundary with the skeleton curve.

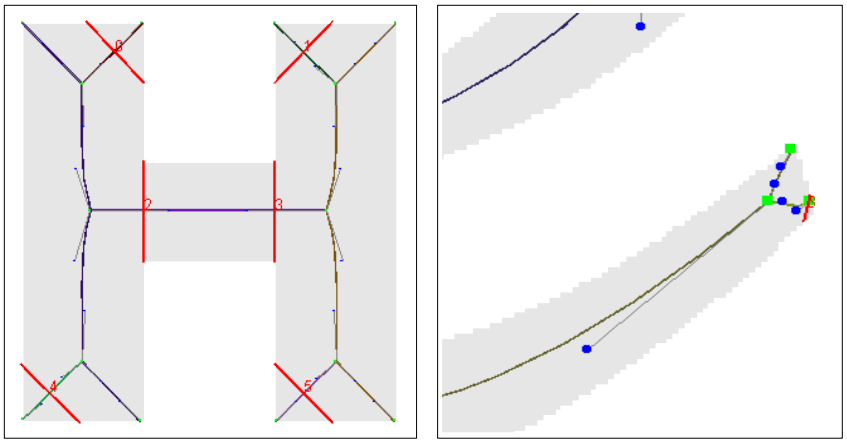


Figure 4.10: Example of shapes with an inaccurate segmentation.

branch point looking for the end points that have a branch point in common. Most of branch points are of degree three, that is three branches emanate from them. As we just mentioned, we need to find the two shortest ones to be removed. Besides the length condition, they also need to satisfy the same pruning condition than the one described in Section 3.1.4, see expression 3.5, here the factor s is $\sqrt{2}$. Figure 4.11 shows an example of this step. We consider the maximal disk centered at the branch point and its radius. If $|ep - bp|_2 \leq \sqrt{2} * r$, the branches will be removed, see Figure 4.12.

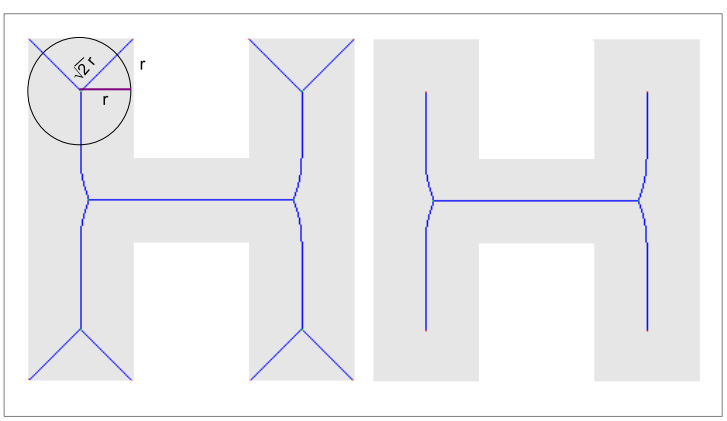


Figure 4.11: Example of squared-ends shape pruning. $Left$, Final Skeleton and selection of $s = \sqrt{2}$. $Right$, Skeleton after extra pruning

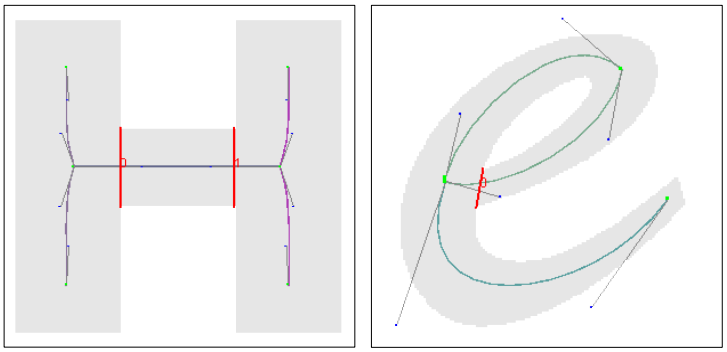


Figure 4.12: Example of shapes with their segment boundaries after pruning.

After this extra pruning step there are still few cases where some of the unwanted branches are not removed. This could be fixed by changing the threshold used in this process but doing this produce the removal of some branches we do not want to remove in cases where we are obtaining desirable results. See Figure 4.13 where the branches at the squared-ends are not being removed resulting in a undesirable shape segmentation.

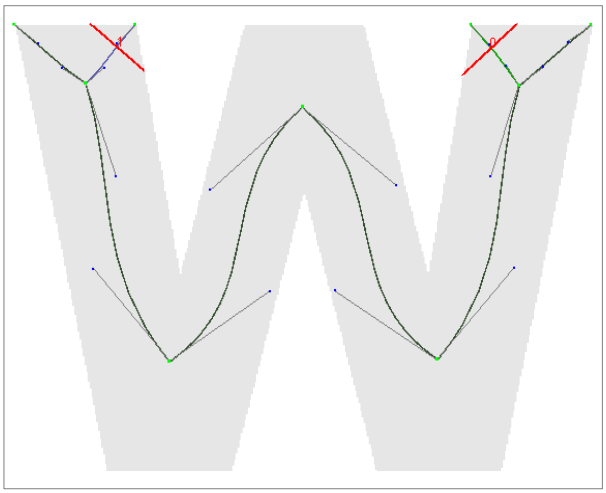


Figure 4.13: Example of shape where the branches are not removed after the extra pruning step.

The implementation for the skeleton computation allows the user the adjustment of the pruning parameter, s (Chapter 3). Therefore, by changing this parameter, the unwanted branches can be removed from these particular shapes without altering all the results. The values for s were in the interval [1.2, 1.4]. Figure 4.14 shows the results obtained after changing the threshold for the pruning step.

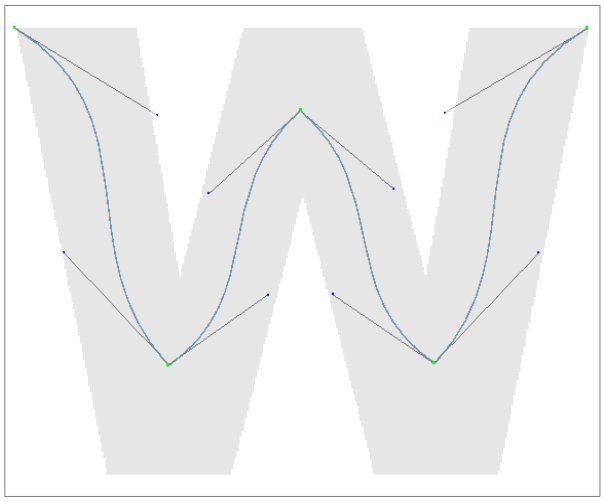


Figure 4.14: Branches are removed after changing the pruning threshold during the skeleton computation.

4.3.2 Multiple Branch Points

Another case is when there are several branch points very close to each other due to the shape protrusions and they are joined by short curve segments. For a illustration an example of this type of configurations is shown in Figures 4.15 and 4.16. When we have this kind of scenarios, if the placement of segment boundaries is done by means of the method previously described (general method), the results are not satisfying. Figure 4.17 is an example of the resulting segmentation. As can be observed, the ideal result would be to segment each protruding part of the shape. Hence, our first approach was to add a segment boundary line to each curve without taking into account the short segment between the branch points and the continuity of the curves as we did in the algorithm explained in the last section.

The first step was to identify the cases where the scenarios described above occurred: For each branch point *bp* we compute the maximal disk centered at it and test which of the other branch points lies inside the disk. With this data we fill a two-dimensional table of relationships. For example, assume that we have three branch points very close each other, then the table would be as shown in Figure 4.18 where every *bp* lies inside the maximal disk of the others thus, the chart is densely filled.

For a completely dense table, we store each curve with its corresponding branch point, then, we find the feature transform of its eight neighbors and send all these parameters to the $getSL(\ )$ method which remains the same. This approach worked for shapes like

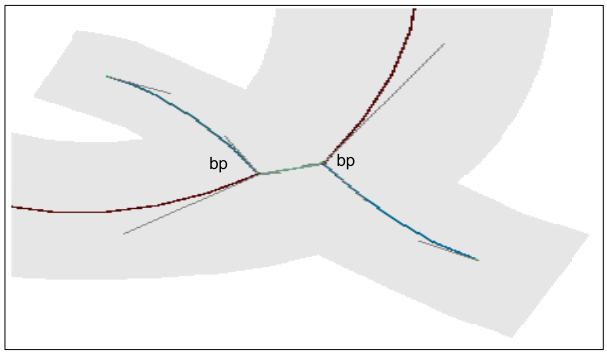


Figure 4.15: Example of a shape whose branch points are joined by a short curve segment.

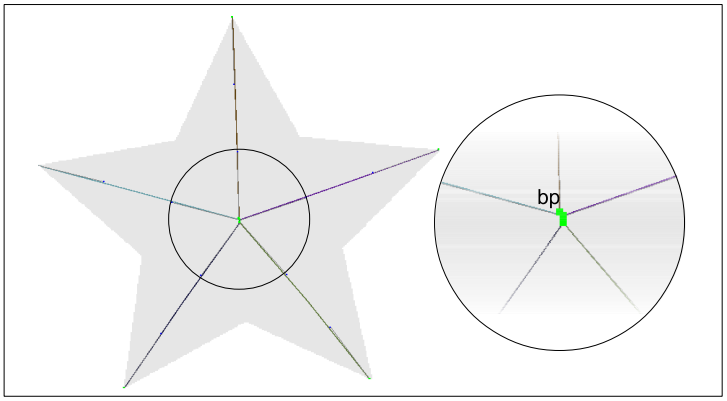


Figure 4.16: The star shape has all its branch points right next to each other.

the one in Figure 4.16 where all the branch points are contained in a small area. The results are show in Figure 4.19, on the left we show a star shape with a more desirable segmentation. However, it is now arguably over segmented and we may want to remove two segment boundaries to create a continuous segment with the surrounding O shape.

The second step is therefore, to decide which segment boundaries should be removed. This situation occurs when the number of branch points is equal to two and the process for removal is as follows: besides computing the segment boundaries and associating each of them with this corresponding *bp*, we also compute the intersection point of the lines with the skeleton curve. As it can be observed, two curves have the same branch point in common, then we compute the distance from this point to the intersection point *ip* on each of its corresponding curves and store the line at the largest distance. The above is done for each branch point.

We pick one of the farthest lines arbitrarily and add it to the list of segment boundaries, then we compare it to the remaining lines by means of the dot product to find the

line that is more likely to be parallel to the previously chosen one. When this parallel line is found we add it to the list. If the number of branch points is bigger than 2 the step for removing segment boundaries is not carried out. See Figure 4.20 where we present the final result.

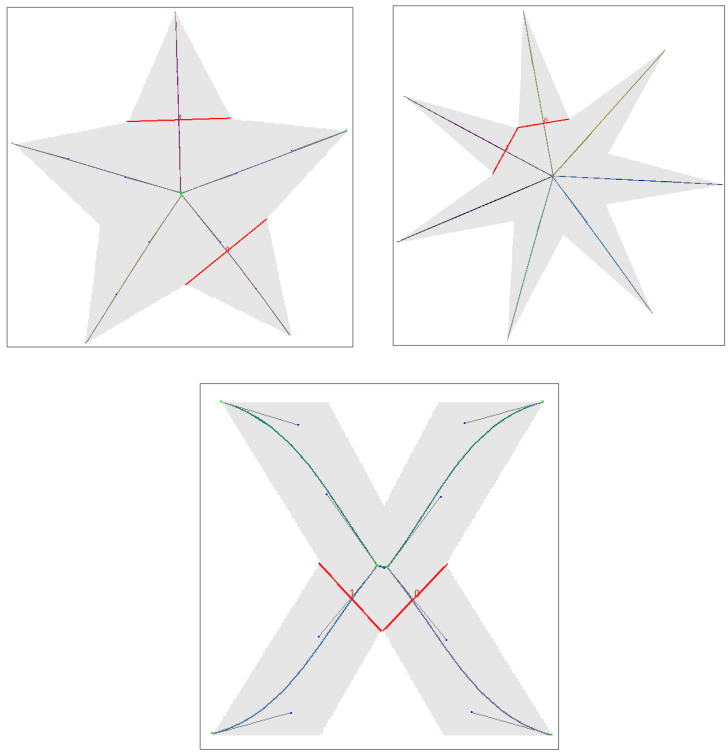


Figure 4.17: Segmentation without taking into account the multiple branch points scenario.

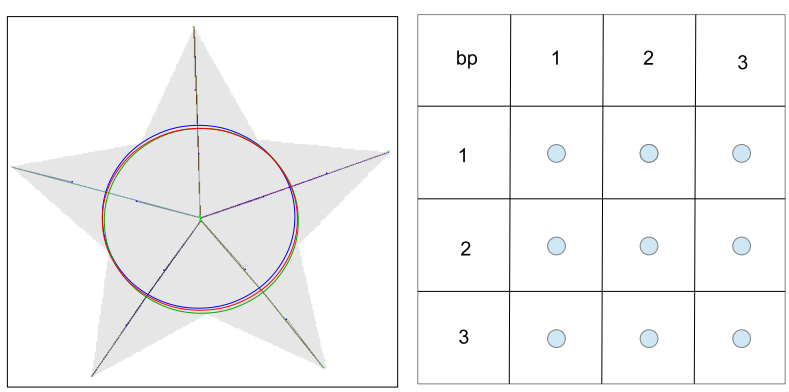


Figure 4.18: Each branch point lies inside the maximal disk centered at the other branch points. The colored circles represent the disks.

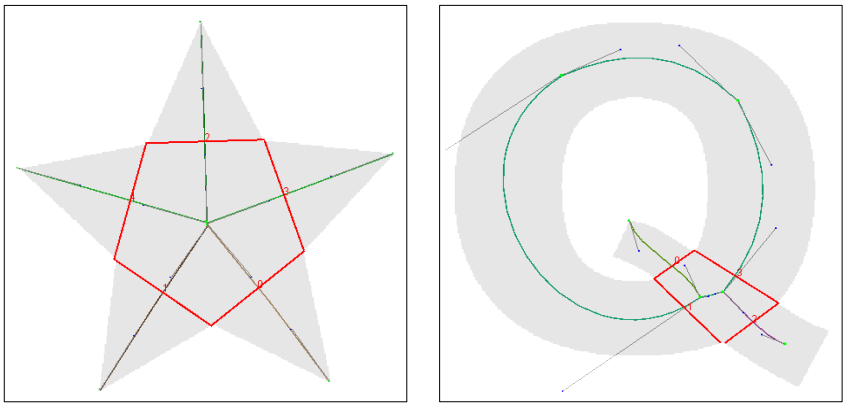


Figure 4.19: Results of segmentation for shapes with branch points close to each other.

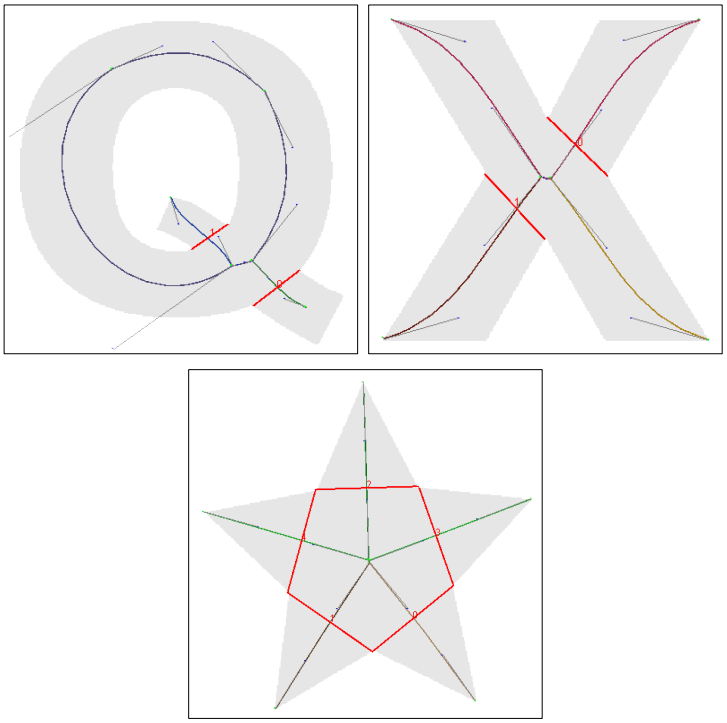


Figure 4.20: Result after removing some segment boundaries, leaving those that are close to be parallel.

There exist cases where not all the shape's branch points are in each others medial disks, this occurs only in certain regions of the shape. For these scenarios, we used a similar chart than in Figure 4.18 but in this case, the chart is partially filled because of the above mentioned. Figure 4.21 is an example of this type of shapes. We identified the

regions with several branch points close to each other (e.g., red circles) and the regions where there is only one branch point (e.g., blue circles). The branch points in each red circle were handled as a single branch point (as previously described for letter Q and X of Figure 4.20) and the branch points in the blue circles were processed with the general segmentation method which is based on the continuity of the skeleton curves at their corresponding branch point. The segmentation results for the shape in Figure 4.21 are shown in Figure 6.22

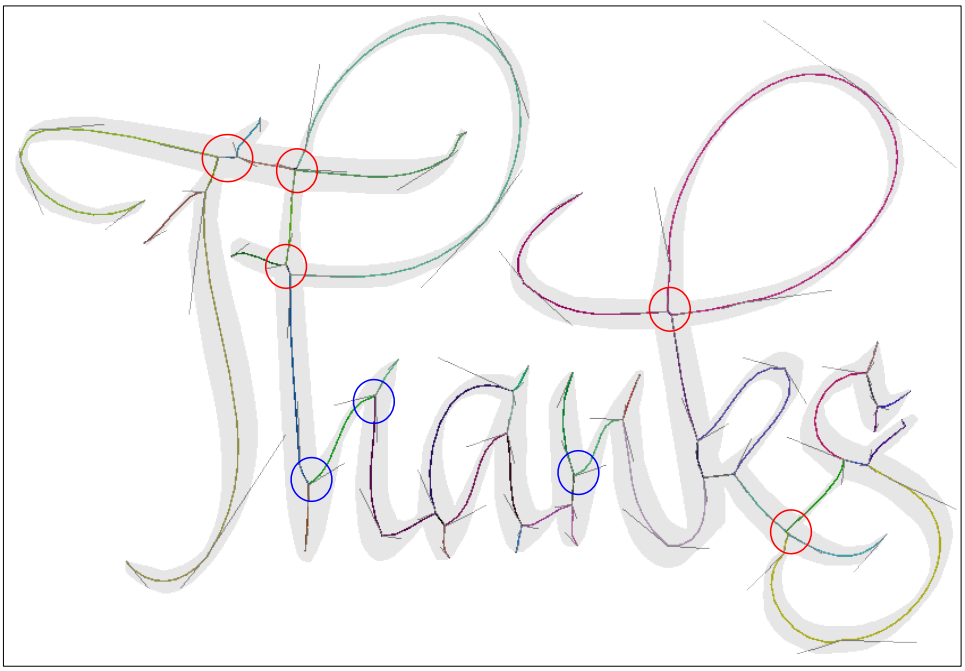


Figure 4.21: Example of a shape with multiple branch points close to each other but only in some regions of the shape (e.g., red circles). The blue circles point out some regions with a single branch point. The chart for this type of shapes is partially filled.

4.4 Intersections

After finding the segment boundaries, we need to extract some information from them, for example, the intersection points of the lines with the boundary curves, the value of the parameter t at the intersection. For each line end point we find the closest curve with the method that will be described in this chapter. Using this method we not only obtain the curve but the closest point on the curve (intersection point) and the value for t. For this purpose every boundary curve and segmenting line has been assigned an ID and two data structures were created, the first one is for the end points and stores the curve ID and the value of t at the intersection point. The second one stores the two end points and the segmenting line ID. Finally, we use a map that relates each element of the intersections data structure with its corresponding segmenting line. Figure 4.22 shows the pseudocode for the intersections.

Output: Map(SLIntersection, SLine)
id=0
for each $sl \in Lines$ **do**
  $b1 = closestCurve(sl.P1)$
  $b2 = closestCurve(sl.P2)$
  $curveId1 = (b1.c).getId$
  $curveId2 = (b2.c).getId$
  $endP1 = EndPoint(curveId1, b1.t)$
  $endP2 = EndPoint(curveId2, b2.t)$
  $Intersection.add(endP1, endP2, id)$
  $Map.add(Intersection, sl)$
  $id = id + 1$
end for

Figure 4.22: Algorithm to compute the intersection of the Segmenting Lines with the boundary curves ($computeIntersections(\ )$ method). It uses two data structures: EndPoint and Intersection.

4.4.1 Closest Curve

This method is based on the algorithm described in Chapter 3 to find a point on a curve. Given a set of boundary curves, we loop over them and use the current curve c_i and an arbitrary point q as parameters for the method point on the curve described in Chapter 3. Recall this method computes the distance between the query point and the possible solutions until the shortest distance is found. We use this distance in order to find the closest curve to q. Once this is done, the method returns the curve, the closest point on it and the value of t, see Figure 4.23.

Input: An arbitrary point q, set of boundary curves bCurves
Output: Closest curve c, closest point p, value of t
$closest = null$
for each $c \in bCurves$ **do**
  $current = pointOnCurve(c, q)$
  if $current.dist < closest.dist$ **then**
    $closest = current$
    $b = c$
  end if
end for
return$(b, closest.P, closest.t)$

Figure 4.23: Algorithm to find the closest curve to an arbitrary point. It returns the curve, the closest point on the curve and the value of the parameter t.

One of the disadvantages of this method is that it slows the process for computing the intersections of the segment boundaries (SL) with the boundary curves, particularly when the shapes contains a great number of branch points (there is a segmenting line per branch point). Currently, for each endpoint of every SL, we loop over all the shape curves until we find the closest one. Having n boundary curves, the intersections are found in $O(n^2)$ time. One option may be the use of an acceleration data structure, a k-d tree, for instance. The idea is to build a k-d tree for the bounding boxes of the boundary curves and implement a ray tracing algorithm to find the intersections [17]. Figure 4.24 shows an example of a k-d tree.

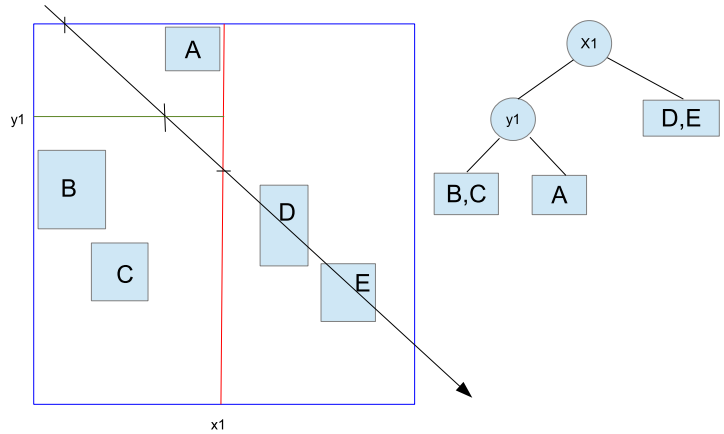


Figure 4.24: Example of a k-d tree and ray tracing to compute intersections.

The space is recursively divided into subspaces until a splitting criteria is satisfied. This criteria could be that a subspace contains a minimal number of objects or the tree has a maximal depth. After the k-d tree has been built, it is necessary to implement a traversal algorithm. For this we need a ray segment and the root of three. If the root is a leaf, then all the objects in the leaf will be intersected and we need to find the closest intersection point to the origin of the ray. If the root is a tree node, we need to determine if the ray lies on one side of the plane or on both sides. In the example above, the ray traverses the left and right subspaces. It also traverses the subspaces delimited by $y1$ but it does not intersects any object. Then, we search for intersections on the right subspace, it intersects two objects so we find the closest intersection point to the origin of the ray. Some objects can be part of more than one subspaces, the use of flags can help to avoid processing an object twice or more times. Using this kind of techniques the computation time for the intersections could be bounded to $O(nlogn)$ time.

4.5 Discussion

In Chapter 2 we reviewed some literature related to shape segmentation based on its skeleton. As in some of the algorithms described in the literature, our goal is to use information from the shape skeleton and the boundary to identify two points on the boundary (on different sides of the shape) to define a line segment that segments the shape.

We can compare some of our results with the results shown by Shah [59]. This approach decomposes shapes by finding protrusions but in some cases the shapes are

segmented in the inside and outside. Our segmentation technique offers acceptable results by segmenting all the protruding parts.

Our approach can also be compared with the one proposed by Zeng et al. [67] where they identify the boundary points, end points, junction points and tangent points in order to find two boundary points to form a "straight connection" between them to segment a shape. The difference between their method and ours is that they look for points on the boundary with negative curvature which is not considered in our technique. Figure 4.25 shows some results obtained for shapes from the MPEG-7 dataset using our segmentation technique.

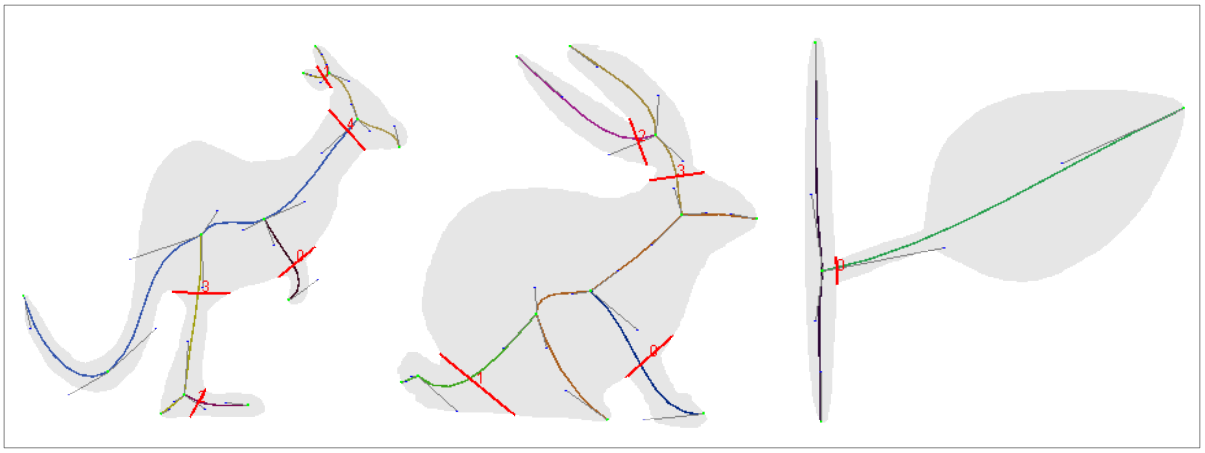


Figure 4.25: Segmentation results for some images from the MPEG-7 dataset.

Our technique segments only one of the ears of both the bunny and the kangaroo. In the case of the kangaroo, the body is not segmented as in the results by Zeng [67]. The main differences are in the kangaroo's tail and the main part of the leaf which are not segmented by our method.

4.6 Summary

We presented our shape segmentation in this chapter. We described the algorithms to find segment boundary line segments defined by two boundary points that allow us to divide a shape into several regions. We described the process to select the aforementioned points, which takes into account the branch points, the curves that emanate from them and the feature transform. We also discussed some cases where the general algorithm for computing the segment boundaries does not gives satisfactory results. For example when

the input image has squared ends an over segmentation is produced. We implemented an skeleton extra pruning step to avoid extraneous segment boundaries. Another issue occurs when there are two or more branch points close to each other, in this case we placed a segment boundary on each curve emanating from the branch point. This solves issues with overlapping medial disks of branch points, e.g., in star shapes. However, it also causes an over segmentation and some lines should be removed. During the computation process we made use of several algorithms for the intersections of the segment boundary lines with the skeleton curves and the boundary curves which were also described in this chapter.

Chapter 5

Direction Lines

In this chapter we will discuss the placement of *Direction Lines* on the different regions generated following the process described in Chapter 4. In Section 5.1 we will describe the concept of Direction Lines and their importance in our research. The process to define and place these lines will be described in Section 5.2. Section 5.3 contains a description of direction line that intersect the boundary segments. In Section 5.4 we will discuss some special cases where the computation of direction line segments follows a different process. After the Direction Lines Segments have been found it is important to compute their intersection with the boundary curves or with the segment boundaries and will be described in Section 5.5. A summary is given in Section 5.6

5.1 Direction Lines

In our work we aim to implement different techniques that can be applied for the automation of *satin-stitch* embroidery designs based on the skeleton of the input shape. Given a shape or design, the automation of embroidery involves its segmentation. The process for shape segmentation was described in Chapter 4. The next step is to decide how the design is going to be embroidered. Previously, we defined the satin-stitch as a series of parallel flat stitches placed close together to give a satin-like finish [40]. In order to create a quality embroidery design the stitches need to be given a direction. This direction is achieved by placing lines on the different regions created by the segment boundaries. We call them *Direction Lines*. Their main objective is to smooth the angle of the stitches. In the next sections we will describe the criteria used for the definition and placement of the direction lines.

5.2 Direction Lines Placement

The skeleton curves represent an important element for the placement of direction lines. We need to identify different characteristics of the curves in order to decide how many lines need to be placed. These characteristics include the length of the curve and if the curve is intersected by a segment boundary or not.

Consider Figure 4.8 that shows an example of a Bezier curve. If a skeleton curve is not intersected by a segment boundary, we need to consider the type of its end points. For example, we place the direction lines differently if one of the endpoints of the curve (P_0 or P_3) is a branch point while the other is a skeleton end point; or, if both of its end points are branch points. In the case where the curves are intersected by a segment boundary, we take into account the intersection point besides the length of the curve. Once these characteristics have been identified, we can specify where to place the direction lines.

Because we are dealing with parametric Bezier curves, the above is done by selecting different values of the curve parameter t. With these values of t we find the direction lines and their intersections with the shape's boundary or with the segment boundaries. There are different scenarios that lead to different types of direction lines. The methods and the different cases will be explained in detail in the next sections. In Figure 5.1 we show an overview of the process to compute the direction lines.

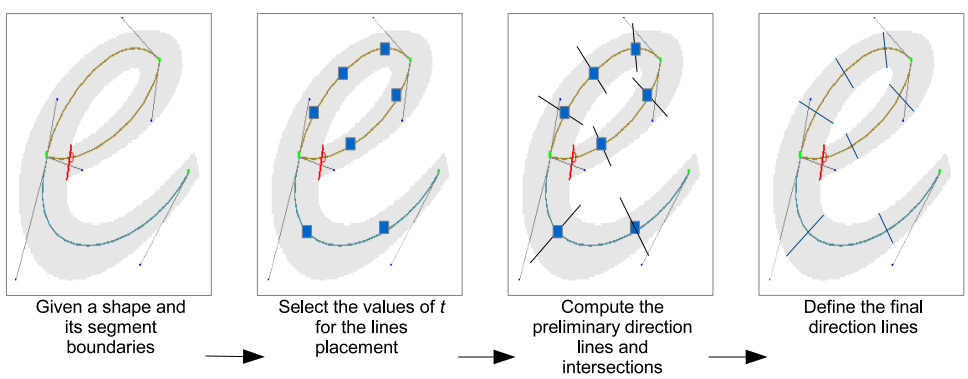


Figure 5.1: Process for the direction lines computation.

5.2.1 Midpoints Computation

The first step in the process for defining the direction lines is to compute the points where the lines will be placed, in other words, the points where the direction lines will

Input: Shape and its segment boundaries.
Output: Direction lines
//Returns the curve and the points at the selected values of t, stored in midPoints
getMidPoints(midPoints);
$dLines = obtainLines(midPoints)$;
//Compute direction lines from shape boundary to segment boundaries
$atSBoundaries = obtainLinesAtSegmentBoundaries()$;
//Compute direction lines at some skeleton curves junctions
$atJunctions = obtainLinesAtJunctions()$;
$computeIntersections()$

Figure 5.2: Algorithm to compute direction lines.

intersect the skeleton curves. As we mentioned above, we need to calculate the length of the curve (in pixels) in order to decide how many lines can be placed on a curve. We divided the algorithm for the midpoints calculation into two parts. In the first part, we compute the midpoints for the curves that are not intersected by any segment boundary. If the end points of a curve (P_0 and P_3) are both branch points and the curve is not long enough we only compute the point at t=0.5, otherwise we compute two points, one at t=0.3 and the other at t=0.7. If the curve has only one end point equal to a branch point and it is not too long, we need to consider the direction of the curve. If the value of t at the branch point is t=0, we compute the point at t=0.3 and if it is t=1, we compute the point at t=0.7. If the curve is long enough, we compute two points one at t=0.3 and the other at t=0.7. Finally, when the curve's end points are both not branch points, we compute a point at t=0.5. If the curve is long enough, we "divide" the curve into four by computing further points at t=0.2 and t=0.8.

In the second part we compute the points for the curves intersected by the segment boundaries. In this case we also consider the direction of the curve and the value of t at the intersection point which was calculated during the computation of the segment boundaries. We define the direction of a curve by computing the parameter t at its end points (P_0 and P_3) by means of the Point-on-the-Curve algorithm described in Chapter 3, which given an arbitrary point returns its closest point on the curve and value of the parameter t at that point. For curves with the shortest length we first identify the direction of the curve. If the value of t at the intersection point of the segment boundary

with the skeleton curve is closer to $t = 0$, we look for a point close to the opposite end at $t = 0.7$. For a segment boundary closer to t=1.0, we use $t = 0.3$. When the curve is longer, we compute the range of values in the curve segment between the value of t at the intersection and $t = 0$ or $t = 1$, accordingly. We divide the complete curve into four but selecting only the points for the values of t within this segment.

The decisions for a curve to be long enough or not and the selection of the different values of t (e.g., $t = 0.3$ and $t = 0.7$) were made empirically. Once these have been specified, we need to compute the points at the desired values of t. The pseudocode for the computation of the points for the direction lines placement is shown in Figure 5.3 and 5.4.

5.2.2 De Casteljau's Algorithm

In the algorithm described above we use the method $computePoint(c, t)$, to compute the point on the skeleton curve at the specified value of the parameter t. This was done by means of the well know De Casteljau's algorithm [9] which is a recursive algorithm for the subdivision of Bezier curves and works for any t within $[0, 1]$. Consider a parametric cubic Bezier curve with its four control points P_0, P_1, P_2, P_3 and the line segments defined by those: $\overline{P_0, P_1}$, $\overline{P_1, P_2}$ and $\overline{P_2, P_3}$ (see Figure 5.5). The line segments are divided in the rate $1 - t$. If we consider t = 0.5, the subdivision of the line segments will generate the points P_{11}, P_{21} and P_{31} and two new line segments. This process is repeated until we obtain the point P_{13} which represents the point on the Bezier curve at t = 0.5. The equations for the computation of a point on the curve at an arbitrary value of t are:

$$\begin{aligned}
P_{11} &= (1-t)P_0 + t * P_1 \\
P_{21} &= (1-t)P_1 + t * P_2 \\
P_{31} &= (1-t)P_2 + t * P_3 \\
P_{12} &= (1-t)P_{11} + t * P_{21} \\
P_{22} &= (1-t)P_{21} + t * P_{31} \\
P_{13} &= (1-t)P_{12} + t * P_{22}
\end{aligned} \tag{5.1}$$

```
//Computing midpoints for curves that are not intersected
//by a segment boundary
for each skeleton curve c do
    lengthC = curveLength(c)
    P0 = c.P0
    P1 = c.P3
    if c is not intersected then
        //Computation midpoints for curves where both endpoints are branch points
        if P0 and P1 = branch point then
            if lengthC ≤ 60 then
                mid = computePoint(c, 0.5)
                midPoints.add(c, mid) //Stores the curve and the point at t=0.5
            else
                mid = computePoint(c, 0.3)
                midPoints.add(c, mid) //Stores the curve and the point at t=0.3
                mid = computePoint(c, 0.7)
                midPoints.add(c, mid) //Stores the curve and the point at t=0.7
            end if
        else if P0 or P1 = branch point then
            //for curves with one endpoint equal to a branch point
            if lengthC ≤ 60 then
                if t = 0 at P0 or P1 then
                    //Compute midpoint at t=0.3
                else
                    //Compute midpoint at t=0.7
                end if
            else
                //Compute midpoints at t=0.3 and t=0.7
            end if
        else
            //for curves with both endpoints not branch point
            if P0 and P1 ≠ branch point then
                if lengthC ≤ 60 then
                    //Compute midpoint at t=0.5
                else
                    //Compute midpoints at t=0.2, 0.5 and 0.8
                end if
            end if
        end if
    end if
end for
```

Figure 5.3: Algorithm to compute midpoints for curves that are not intersected by segment boundaries.

```
for each skeleton curve c do
    if c is intersected by one segment boundary then
        if lengthC ≤ 40 then
            //tDirection computes the value of parameter t at curve points P_0 and P_1
            t= value for segment boundary intersection
            t_0 = abs(tDirection(c, P_0) − t) //absolute value
            t_1 = abs(tDirection(c, P_1) − t)
            if t_0 < t_1 then //t is closer to P_0
                if tDirection(c, P_0) = 0 then
                    //Compute midpoint at t=0.7
                else
                    //Compute midpoint at t=0.3
                end if
            else
                //We follow the same criterion for (tDirection(c, P_1))
            end if
        else
            if t < 0.5 then
                t_aux = (1 − t)/4
                t_0 = t_aux + t
                t_1 = (3 ∗ t_aux) + t
                //Compute midpoints at t_0 and t_1
            else
                t_aux = t/4
                t_1 = 3 ∗ t
                //Compute midpoints at t_aux and t_1
            end if
        end if
    end if
end for
```

Figure 5.4: Algorithm to compute midpoints for curves intersected by segment boundaries.

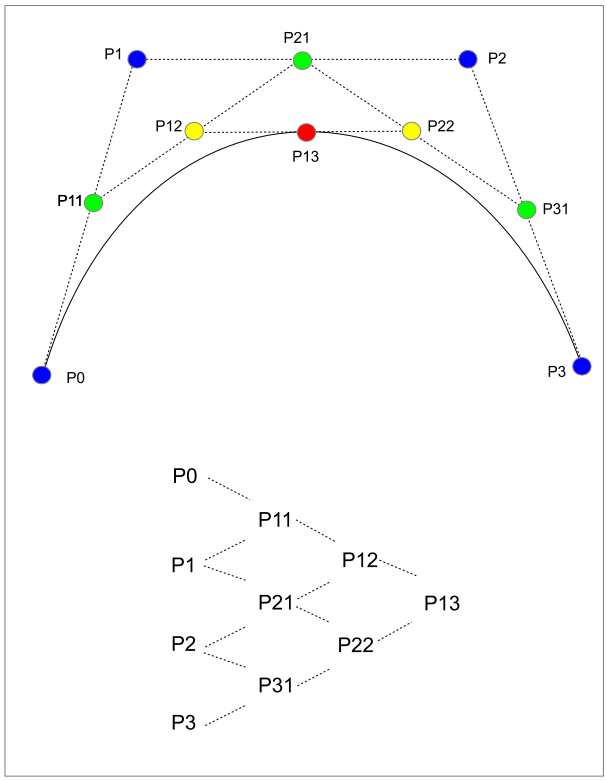


Figure 5.5: Curve subdivision by De Casteljau's algorithm.

The method $computePoint(c, t)$ returns the last three points found, that is P_{12}, P_{22} and P_{13} which are then used to compute the direction lines as we will describe in the next section.

5.2.3 Obtaining Direction Lines

With the data returned from the *computePoint* method we can find the direction lines that define the angle of the stitches in the embroidery design. The information we have to do this, is the point on a curve at a specified value of the parameter t and the last line segment (P_{12}, P_{22}) obtained from the de Casteljau's algorithm. Every direction line is based on two line segments, one from the point on the curve to one side of the skeleton and the other from the point on the curve to the other side. In order to obtain these two line segments, we compute a line segment orthogonal to (P_{12}, P_{22}), which is then rotated to obtain the second one. These orthogonal lines segments are translated to the point P_{13} (the point at t) and then made long enough such that they intersect a boundary curve, see Figure 5.6.

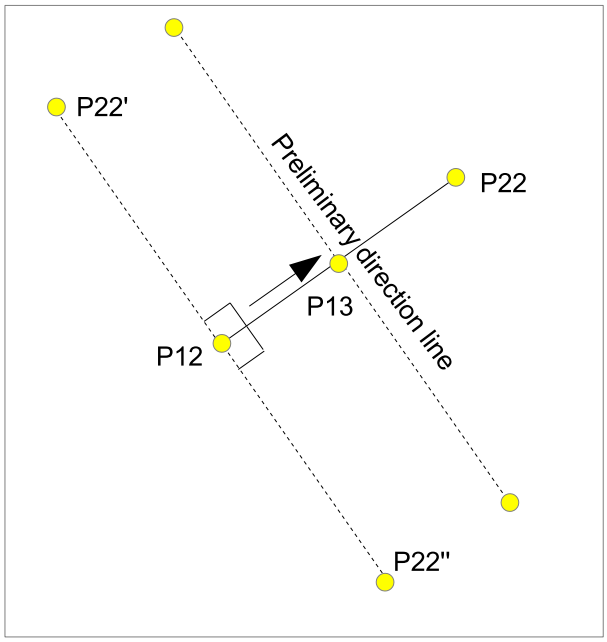


Figure 5.6: Obtaining the preliminary direction line segments for t=0.5.

Our initial approach to make the line segments longer was to compute the radius of the shape at the point P_{13} by using the feature transform and a scale factor. However, because of the direction of the line segment, the boundary may not be reached. We also observed that some direction line segments intersected each other (see Figure 5.7). Therefore, we extrapolate the line segments to the boundary with the help of the feature transform. Using the feature transform has the major advantage that the extrapolated line segments do not intersect.

In the algorithm of Figure 5.8 we have as input a preliminary direction line segment, then, we find the feature transform of one of its end points (the one on the skeleton), storing it in ep and we define a new line segment with P_1 and ep. Normalized direction vectors for both line segments are calculated and used to compute the dot product between them. If the dot product is less than a threshold, it means that the vectors are not parallel and that the feature transform point ep is in another direction. For non-parallel segments, we step towards the boundary by defining a new ep and computing its feature transform $ep2$. With this new end point we define a vector $P_1, ep2$ and compute the dot product again. We repeat the process until the vectors are sufficiently parallel and we obtain a point in the boundary. This method returns a preliminary direction line segment for all direction lines.

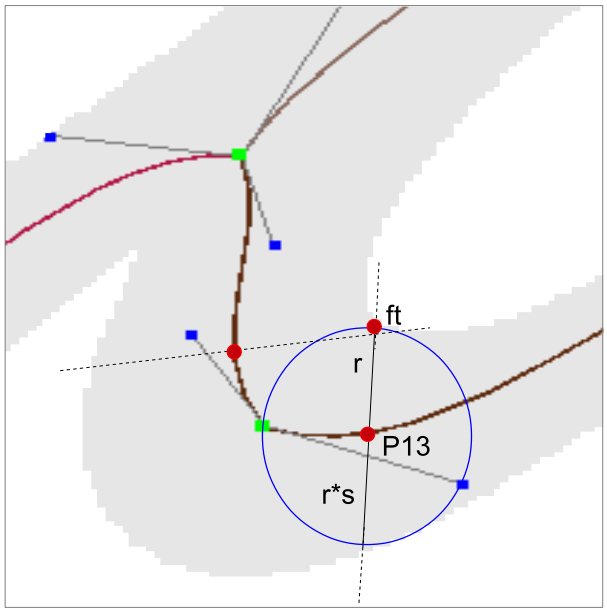


Figure 5.7: Obtaining longer direction line segments. The radius r of the disk centered at P_{13} is computed and scaled by s. Some direction lines may intersect.

5.3 Direction Lines at Segment Boundaries

In the automatic embroidery process it is important to give the stitches a smooth flow, which is achieved with the placement of the direction lines. Sometimes, the lines described in the last section are not enough but the results can be improved by placing more direction lines. For example, in places where there may be a noticeable change of direction which occurs in regions enclosing the branch points. It can be observed (see Figure 5.1) that in these regions we also find a segment boundary that can be used to place the new direction line segments and as described in Section 5.2.3 they are defined by a pair of line segments. The idea is to define one line segment with one endpoint on the boundary and the other one with one end point on a segment boundary. We used the branch points of the skeleton as reference for the computation of the segment boundaries. We made use of a map that relates each branch point with a segment boundary and another map that relates the segment boundary with boundary points. We use these maps to compute the direction line segments as well. For each segment boundary we need to compute a pair of line segments to define a direction line. First, we compute the midpoint of the segment boundary (sb), then, we loop over the boundary points (feature transform points, ftp) and create a new line segment from the current ftp to its related

```
Input:  Direction line segment l, feature transform ft
Output:  Elongated direction line
P1 = Direction line end point on the skeleton
ep = ft[P1] //closest point on boundary to P1
ln = normalize(l) //normalized direction vector of l
laux = line(P1, ep) //line segment from P1 to ep
ln2 = normalize(laux)
dot = ln · ln2
//Check if the two vectors are parallel enough
if dot < parallel_threshold then
    ep = P1
    while ep is not boundary do
        ep = ep + ln //step towards shape's boundary
        ep2 = ft[ep]
        laux = line(P1, ep2)
        ln2 = normalize(laux)
        dot = ln · ln2
        if dot > parallel_threshold then //Vectors are parallel enough
            ep = ep2
        end if
    end while
end if
eLine = line(P1, ep)
```

Figure 5.8: Algorithm to obtain larger preliminary direction lines.

branch point (bp). We are looking for the line segment that is close to orthogonal to the segment boundary, and once we find such a line segment, we select the ftp as the endpoint for the new direction line. The second line segment is defined by the branch point (bp) and the midpoint on the segment boundary (sb). The algorithm to find the direction line segments at segment boundaries is shown in Figure 5.9.

5.4 Special Cases

As well as in the computation of the linear segment boundaries (see Section 4.3), there are some special cases where the direction lines need to be defined in a different way. One of these cases is when we have small shapes, spheres, or shape that are close to a

```
Input:  List of branch points, related ft points and segment boundaries
Output:  Direction line segments at segments boundaries
for each bp and its sb do
    //ftp = List of feature transform points related to the current branch point bp
    ln = normalize(sb)          //sb = segment boundary
    pm = middle_point(sb)                    //sb midpoint
    temp = MAX_VALUE
    for each point p in ftp do
        laux = line(bp, p)
        ln2 = normalize(laux)
        dot = ln · ln2
        angle = acos(dot)
        dif = 90 − angle
        if dif < temp then
            temp = dif
            fp = p
        end if
    end for
    l1 = line(bp, fp)
    l2 = line(bp, pm)
end for
```

Figure 5.9: Algorithm to compute the direction line segments at segment boundaries.

square. The skeleton for these kind of shapes is either a single point or reduced to a point after pruning. During pruning for a point to be part of the skeleton the feature transform points must be on different curves of the approximated contour. But in these cases the feature transform is on the same boundary curve for the current point and its neighbor.

Our approach to solve this problem is to compute the middle point of the input binary shapes without a usable skeleton after pruning. Once the midpoint is found, we verify that it is not part of the boundary. If it is not part of the boundary, we use this point as the midpoint to compute the direction line segments. If the midpoint is outside the shape, we look for the maximum and minimum position of pixels inside the shape and use them to compute the midpoint (mp) we are looking for. In either case, the second step is to compute the direction lines segments. We find the feature transform point fp of mp, with this two points we define a line segment (fp, mp). The second line segment is

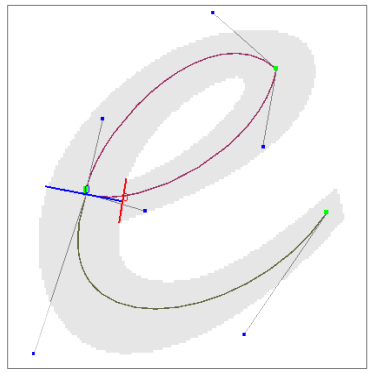


Figure 5.10: Example of a shape with its segment boundary (red) and the direction line segment (blue).

found by rotating (fp, mp) by 180°. Finally the line segments are extrapolated following the method described in Section 5.2.3. Figure 5.11 describes the algorithm to compute the direction line segments for small shapes with no skeleton and Figure 5.12 shows an example of this type of shapes.

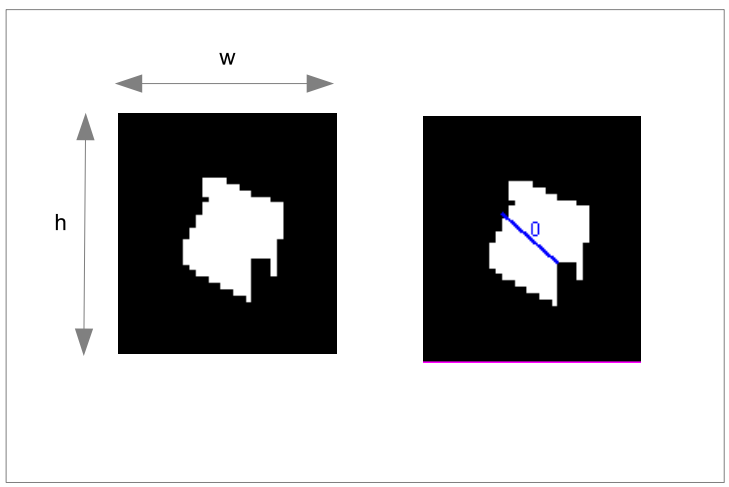


Figure 5.12: Small shape with no skeleton and its direction line segments.

Another case where we need to add extra direction line segments is when we have shapes with sharp corners. The direction line segments are needed to smooth the angle of the satin-stitch and avoid gaps in the design when it is being embroidered. When we have shapes with the sharp corners then their skeleton also has a sharp corner and during curve approximation the skeleton curve is split. In this case, we place direction line segments at this joint point between the two Bezier curve segments. We look for the

```
Input:  binary shape of size w × h with no skeleton
Output:  shape's midpoint mp
if skeleton = null then
    mw = wdiv2
    mh = wdiv2
    p = Point(mw, mh)
    if p is boundary then
        dx = ((maxX − minX)/2) + minX
        dy = ((maxY − minY)/2) + minY //min and max positions of white pixels
        mp = Point(dx, dy)
    else
        mp = p
    end if
end if

//In the second part we compute the direction line segment
Input:  small shape midpoint mp
Output:  direction line segments
fp = ft[mp]
l1 = line(mp, fp)
l2 = Rotate180(l1)
```

Figure 5.11: Algorithm to compute direction line segments for small shapes with no skeleton.

curves with at least one endpoint (P_0 or P_3) different than a branch point and obtain the tangent line segment at that point. Once this line segments have been obtained, they are organized by the endpoint they share. The tangent line segments are normalized and the dot product between them is computed. If the dot product is less than 0.55 (the tangent line segments are approximately in the same direction) then they are averaged and extrapolated to the boundary in order to obtain the new direction line segments. Figure 5.13 shows some examples of the case just described.

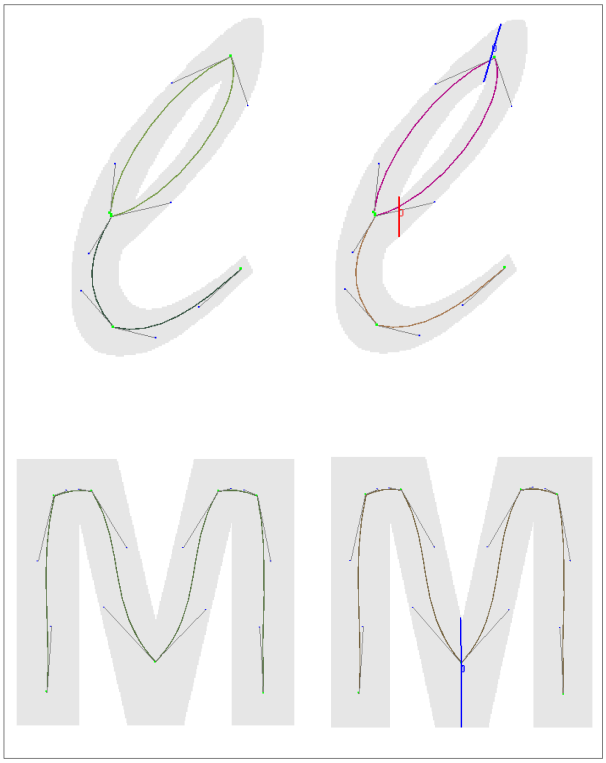


Figure 5.13: Examples of shapes where direction line segments (blue) were placed at some skeleton curve junctions.

5.5 Intersections

The direction line segments computed in the sections above are considered the preliminary. The final direction line segments are defined by the intersection points of the line segments with the continuous shape boundary or in some cases with the segment boundaries, as described in Section 5.3. In this section we will discuss the process to compute these intersections. We start by describing the data structure used to store the information obtained from the intersection computations.

The preliminary direction line segments are defined by two line segments that have in common a point on a skeleton curve, however we are interested in the intersection of each segment with the boundary or with the segmenting lines and not with the skeleton curve precisely, that is that the final direction line segments will be defined by a boundary to boundary or by a boundary to segment boundary line segment. Therefore, the first data structure named EndPoint stores the following: curveID (boundary curve intersected, if

it is the case), the value of the parameter t at the intersection point, intersects segment boundary = false/true (we need to specify if the direction line segment intersects the segment boundary or not), segment boundary ID (if it is the case). When the intersections are computed, we also obtain the point p_i. The point p_i is used in the second data structure which is DirectionLine and is formed by the two EndPoint and the line segment defined by the intersection points (p_i). Finally the intersection are stored in a list of elements of type DirectionLine.

In order to compute the intersections (*computeIntersections*) we use several methods, for every pair of preliminary direction lines. The first method used is *findIntersection* which takes the line segment and an endpoint (P_2) as parameters. In this method we first check if the line intersects a segment boundary at all. If no intersection with a segment boundary is found, a line-curve intersection is computed (*findCurveIntersection*) by means of the closest curve method described in Section 4.4.1. The routine returns the intersection point p_i, the curve intersected c_i and the value of the parameter t_i at the intersection. EndPoint stores: the curve ID, t_i and intersect segment boundary = false. *findIntersection* returns to *computeIntersections*: EndPoint and p_i. These elements are stored in DirectionLine data structure as mentioned before. When a line segment intersects a segment boundary, EndPoint stores segment boundary ID, t_i and intersects boundary = true. Its pair segment intersects the shape's boundary therefore EndPoint is defiened as described in the first part. Figure 5.14 contains the pseudocode for the intersections computation and the method to compute the line-line intersection will be described in the next section.

```
Input:  List of preliminary direction line segments (lines)
Output:  List of intersections
for each pair l1,l2 in lines do
    p1 = l1.P2
    p2 = l2.P2
    Point1, EndPoint1 = findIntersection(p1, l1)
    Point2, EndPoint2 = findIntersection(p2, l2)
    //Intersection points define the direction line segment
    dLine = (Point1, Point2)
    directionLine = (EndPoint1, EndPoint2, dLine)
    intersection.add(directionLine)
end for
```

Figure 5.14: Algorithm to compute the direction line segments intersections, returns the final direction line segments.

```
// findIntersection method
Input:  Line segment endpoint p, line segment l
Output:  intersection point and EndPoint
// line-line intersection, SBIntersection contains the
//segment boundary ID and the value of the parameter t
(point, t, SBIntersection)sbI = segmentBoundaryIntersction(l)
if  sbI ≠ null then
    ep = segment_boudaryID, true, t) //ep is of type EndPoint
    return (point, ep)
end if
(point, curve, t)curveI = findCurveIntersection(p)
if curveI ≠ null then
    ep = (curveID, false, t)
    return (point, ep)
end if
```

Figure 5.15: Method to compute intersections, returns the intersection points.

5.5.1 Line-Line Intersection

In the last section we described how we compute the intersections of the direction line segments, some of these intersections are with the shape boundary and some others

with the segment boundaries. In order to compute the latter ones we make use of the algorithm *Intersection Point of two line segments* described by Vince [63]. Assume we have two line segments $\overline{p0, p1}$ and $\overline{p2, p3}$, the intersection point is given by

$$\begin{aligned} p &= p0 + t(p0 - p1) \\ p &= p2 + s(p3 - p2) \end{aligned} \tag{5.2}$$

The parameters t and s are within the interval [0,1]. Then,

$$p0 + t(p1 - p0) = p2 + s(p3 - p2) \tag{5.3}$$

and can be computed as follow

$$s = \frac{(p0 - p2) + t(p1 - p0)}{(p3 - p2)} \tag{5.4}$$

$$t = \frac{(p2 - p0) + s(p3 - p2)}{(p1 - p0)} \tag{5.5}$$

Using 5.5 the value of t is computed for each component x,y

$$\begin{aligned} t &= \frac{(x2 - x0) + s(x3 - x2)}{(x1 - x0)} \\ t &= \frac{(y2 - y0) + s(y3 - y2)}{(y1 - y0)} \end{aligned} \tag{5.6}$$

Leading to

$$s = \frac{x0(y2 - y1) + x1(y0 - y2) + x2(y1 - y0)}{(x1 - x0)(y3 - y2) - (x3 - x2)(y1 - y0)} \tag{5.7}$$

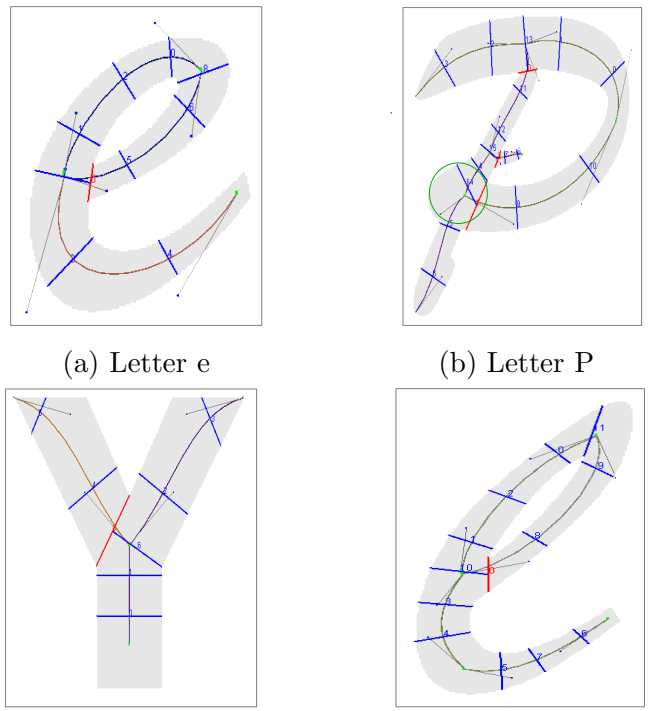
The same is done for t, therefore

$$t = \frac{x0(y3 - y2) + x2(y0 - y3) + x3(y2 - y0)}{(x3 - x2)(y1 - y0) - (x1 - x0)(y3 - y2)} \tag{5.8}$$

Finally, by substituting these values of t and s in the equation 5.2 we obtain the intersection point.

5.5.2 Results

In this section we present some examples of shapes with their segment boundaries and and the different direction line segments after computing the intersections.



(a) Letter e (b) Letter P

(c) Letter Y (d) Letter l

Figure 5.16: Lettering Examples

In Figure 5.16 we present some lettering examples, where Figures 5.16a and 5.16d present all the types of direction segment lines, that is, direction lines on the skeleton curves, at segment boundaries and at some junctions. In all cases, except for Figure 5.16b we observe how the segment lines intersecting the segment boundaries can contribute to smooth the angle of the satin-stitch by changing slightly but this does not happen with letter P, in this case the segment line that intersects the bottom segment boundary has the wrong direction (green circle).

In Figure 5.17 we present a pair of star shapes, these examples are some of the special cases described in Chapter 4 where we treat the set of branch points as a single point and place a segment boundary on each curve. As we can see in these cases not all the segment boundaries are intersected by the direction line segments because we removed some. Placing a direction line segment per segment boundary causes some intersections between

direction line segments. Therefore, after computing all the direction line segments, we detect the intersections between them and remove those lines that generate the highest number of intersections until no intersections remain.

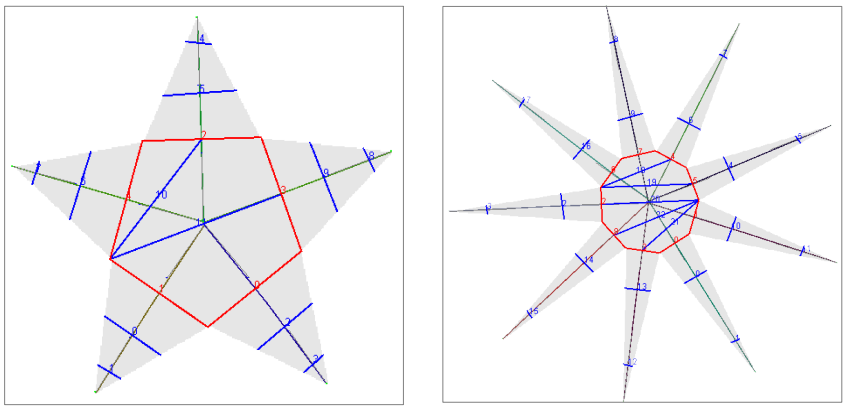


Figure 5.17: Star shapes with segment boundaries and direction line segments.

Figure 5.18 shows a more complex example where most of the lines are well placed and present an acceptable segmentation. Arguably, there is a poorly placed segment boundary caused by a skeleton branch that ends on a corner (green circle).

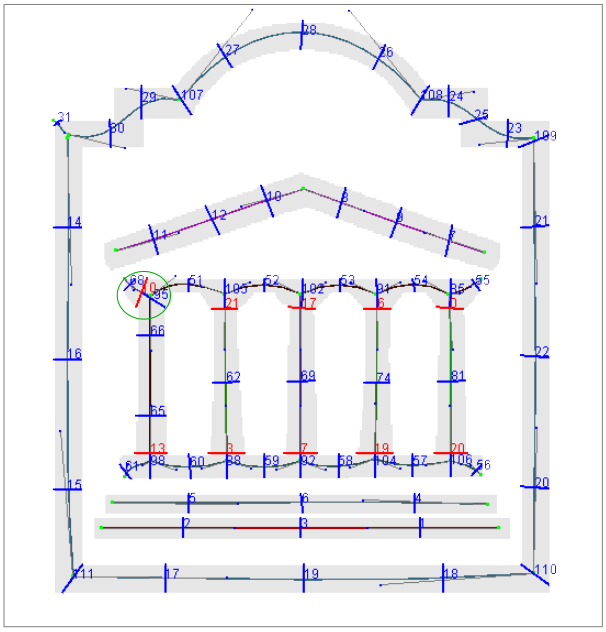


Figure 5.18: uOttawa logo.

5.6 Summary

In this chapter we have discussed the placement of direction line segments. The main objective is to give satin-stitches an angle in order to produced high quality embroidery designs. We described the process to compute points on the skeleton curve at different values of the parameter t. These points are our reference for the placement of the line segments. Once the points have been computed by means of De Casteljau's algorithm, we use the information obtained from it to find preliminary direction line segments. The final line segments are defined by the intersections of the lines with the shape boundary or with a segment boundary. In some cases we added direction line segments at skeleton curve joins in order to improve the results. We also presented some results of the placement of direction segment lines. We showed some segmented lettering examples with their direction line segments and some special cases like star shapes which present multiple branch points. Finally, we presented the direction lines for the University of Ottawa logo as illustration of a more complex example.

Chapter 6

Results

In this chapter we will present and discuss the results obtained for shape segmentation and for the placement of direction lines. In Section 6.1 we will discuss the results obtained for simple shapes (e.g., lettering). Section 6.2 will described the evaluation of special shapes (for example, stars, squares, circles). In Sections 6.3 and 6.4 we will review the results for basic and complex logos, respectively. Finally, in Section 6.5 we summarize our results.

Our research is focused on the development of techniques that could be used in CAD systems for embroidery design, hence, we divided the shapes for the evaluation into several groups: lettering, special shapes, basic and complex logos. Some of the shapes used in this section were obtained from the Open Clip Art Library (images in public domain) and some others were obtained from the commercial software used to test our results[1]. As described in Chapters 4 and 5, when the shapes were processed with our implementation, we assigned an ID to the boundary and skeleton curves as well as to the segment boundaries and direction lines. We also compute the values of the curves parameter, t, at their intersection point with the segment boundaries and direction lines and store them in different data structures. This information was summarized in a single file that is called after loading the design in the embroidery design software telling the software where to place the segment boundaries and the direction lines to control the flow of the stitches.

Embroidery designs are very popular in the textile industry, (e.g., publicity, sports, brand clothes) where the use of logos is an important factor, for this reason we include the logos categories and the testing using the software aforementioned. Our implementation

[1]Software name withheld for confidentiality

was developed in Java and executed on a Dell PC with an Intel Core i7 CPU, 4GB RAM running a 64-bits Windows7 operating system. The processing times are summarized in Table 6.1 and depend on the complexity of the design. The criteria to decide if the design is complex or not is based on the number of branch points of the skeleton and, if the branch points have overlapping maximal disks, and how many of them overlap. In addition, the number of shapes in the design is considered.

The designs in the logos categories consist of several shapes and each of them is processed individually. If the total number of branch points of the design's skeleton is less than 20, the total number of overlapping maximal disks (centered at the branch points of its skeleton) is less or equal to 3 and the number of shapes in the design is less than 25, then we consider the design as a basic logo for the purposes of our evaluation. Otherwise, we consider it as a complex logo.

Table 6.2 presents a description of the errors that we have observed after processing the designs and they will be pointed out in each of the examples in this chapter. The errors can be related to the continuity criterion explained in Chapter 4 for shape segmentation. Error 1 describes a situation where we would expect a segment that connects two opposite skeleton branches but instead a segment that connects two skeleton branches next to each other is produced. Error 2 is a similar error but at T-junctions where we again would expect a segment connecting the two opposite skeleton branches. One reason for the occurrence of these two errors is our use of the tangent at the branch point to decide continuity while in fact a more global criterion would be needed. We have seen some cases where the segment boundaries are not respected during the embroidery process and this we describe as an error number 3. This may be due to the end points of segment boundaries that overlap once loaded in the embroidery software. The exact source of this error is unclear to us. Error 4 refers to rare cases where no direction lines are produced, possibly due to aggressive pruning of the skeleton. Error 5 is produced in shapes where several branch points are very close to each other in different parts of the design, creating multi-way junctions. This often results in either an over-segmentation of the shape or missing segment boundaries. Finally, error number 6 is assigned to direction lines which do not have the expected orientation. This can occur close to segmentation boundaries due to the skeleton shape at branch points.

Category	Processing Time (s)
Lettering	0.8-1.0
Special Shapes	2.0-3.0
Basic Logos	4.0-6.0
Complex Logos	6.5-10.0

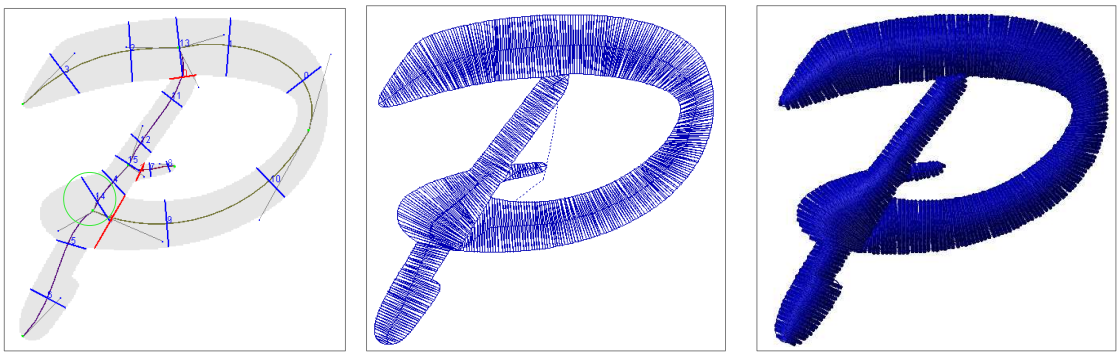
Table 6.1: Processing time by category

Number	Description
1	4-way junction with incorrect continuity
2	Continuity constrains violated in T junctions
3	Intersection between direction lines or segment boundaries
4	Segments without direction lines
5	Overlapping multi-way junction
6	Incorrect orientation of direction lines

Table 6.2: List of observed errors during the processing of embroidery designs

6.1 Lettering

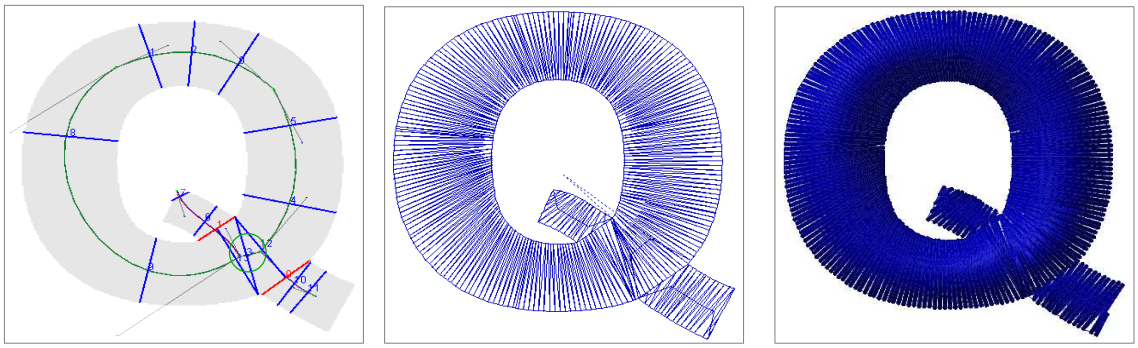
In this first category we will evaluate some lettering examples which are some of the simplest shapes we have processed with our implementation. In most cases, these examples consist of a single shape and we present their skeletons with their segment boundaries and direction lines.



(a) (b) (c)

Figure 6.1: Letter P. a) Segment boundaries and direction lines. b) Test in embroidery design software. c) 3D view of embroidery design.

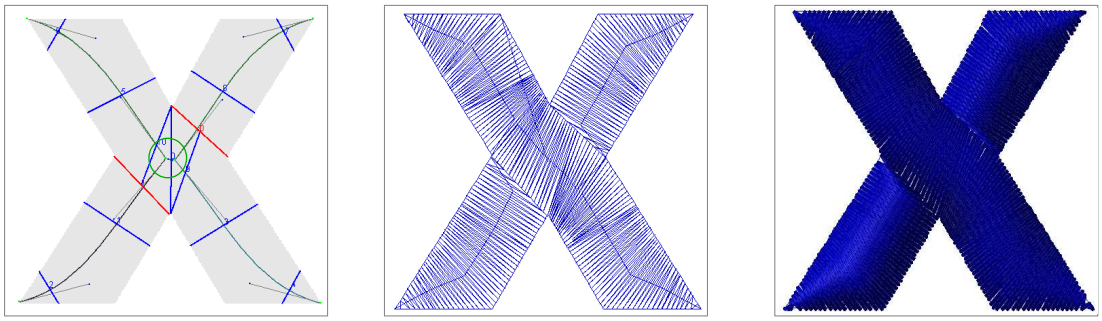
Figure 6.1 shows the segmentation of letter P with its corresponding segment boundaries and direction lines. This shape consists of three branch points with no overlapping maximal disks. The shape presents "T junctions" at its three branch points which are respected by our segmentation algorithm. With the placement of segment boundaries, the shape has been divided into three regions where we placed direction lines. The direction lines allowed a continuous flow of the stitches in each region but the results could be improved by changing the angle of some directions lines, specifically those that intersect the segment boundaries (e.g., Figure 6.1 a) green circle). Figure 6.1 b) and c) simulate how the embroidery would look like.



(a) (b) (c)

Figure 6.2: Letter Q. a) Segment boundaries and direction lines. b) Test in embroidery design software. c) 3D view of embroidery design.

Figures 6.2 and 6.3 are other lettering examples, that belong to one of the special cases discussed in Chapter 4, presenting four way junctions. Both shapes have two branch points and the maximal disks centered at them overlap. The stitching flow follows the direction lines respecting the regions created by the segment boundaries. In c) of both figures we can see that the results were improved after the branch point pre-processing and segment boundaries removal, however they could be further improved by changing the placement of the direction lines in the middle of the skeleton curve between the branch points (green circle).



(a) (b) (c)

Figure 6.3: Letter X. a) Segment boundaries and direction lines. b) Test in embroidery design software. c) Final result, 3D view of embroidery design.

We also evaluated different types of fonts, e.g., Figure 6.4 and Figure 6.5 show font Liberation Serif, lower case and capital letters respectively, where in most of the letters the stitches respect the segment boundaries and flow smoothly through the shape but

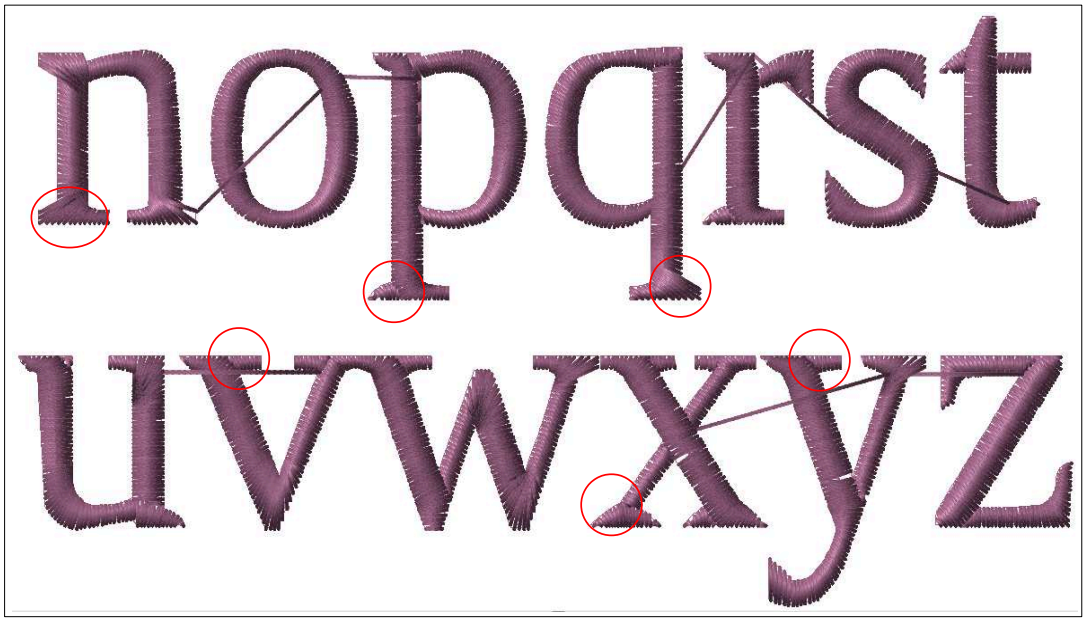
also where the continuity criteria for the placement of segment boundaries does not work as expected at the T-junctions (red circles). Figure 6.6 show some letters with their corresponding segment boundaries and direction lines, where we can see error 2 in letter G and at the bottom of letter m.

Another font type is Purisa of Figures 6.7 and 6.8. In these cases, the stitches follow the direction lines properly and the segmentation resulted as expected in most cases, except for that in letter k of Figure 6.7 (see red circle, error 2), where the vertical part is being segmented at the bottom. An optimal segmentation would be like the one presented in capital letter K of Figure 6.9.

Finally, in font Eufm of Figure 6.10 we see two cases where segment boundaries are missing at the T-junctions, resulting in overlapped stitches and small regions with no filling (e.g., letters n and t, error 4). In letter Z of Figure 6.11, the stitches do not respect the segment boundaries (error 3) and the do not follow the direction lines properly causing the stitches to overlap and gaps. In the rest of the letter in both figures the stitches have a continuous flow presenting optimal results. Figure 6.12 show some letters with their direction lines and segment boundaries, we can see in Figure 6.12 b) that a segment boundary is missing (error 4).

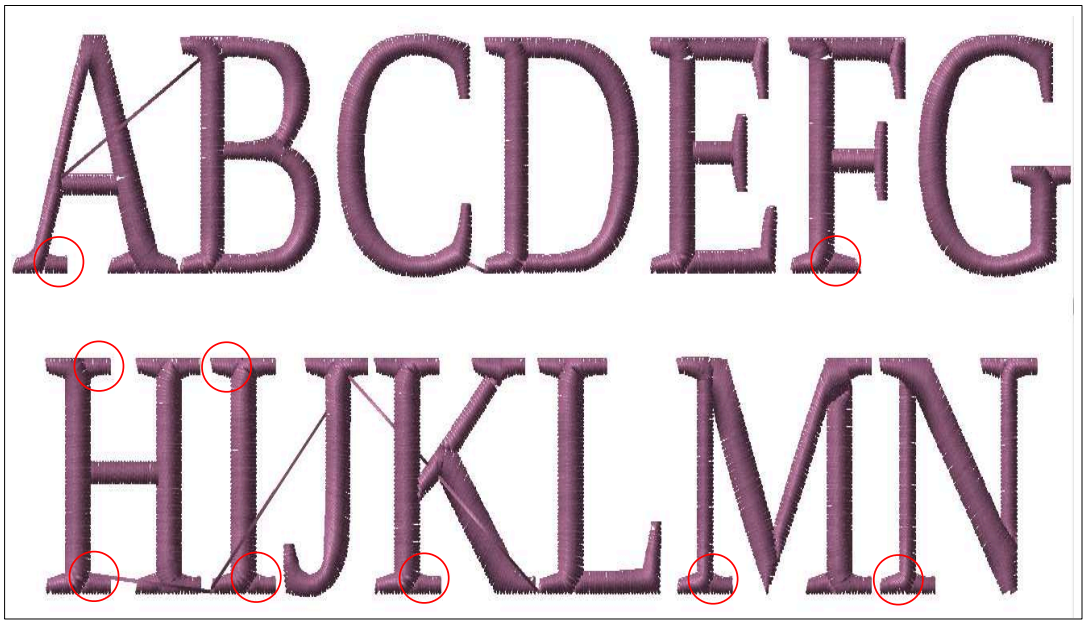
In some cases, as mentioned above, there could be an improvement by changing the angle of some direction lines or even by increasing the number of lines, nonetheless, the lettering examples present satisfactory results in general.

(a)

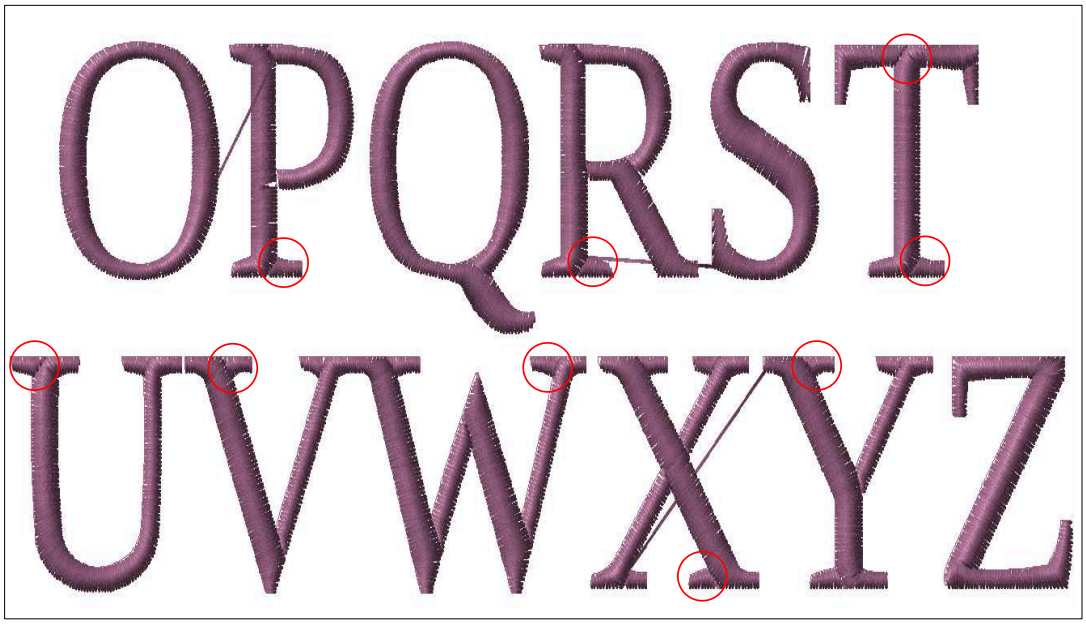


(b)

Figure 6.4: Font Liberation Serif, lower cases. Embroidery design. The red circles show some number 2 errors, and in letter b in (a) we also can see error number 6 (incorrect orientation of direction lines).

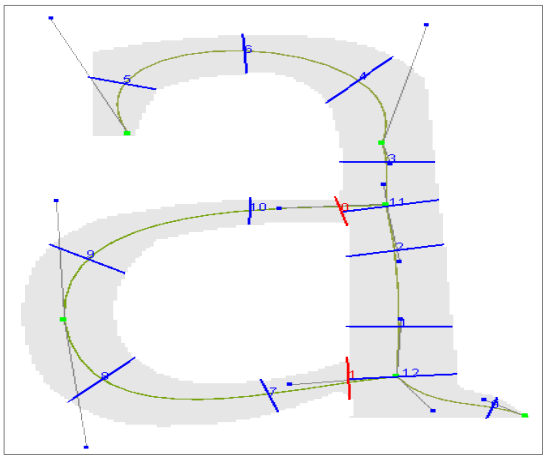


(a)

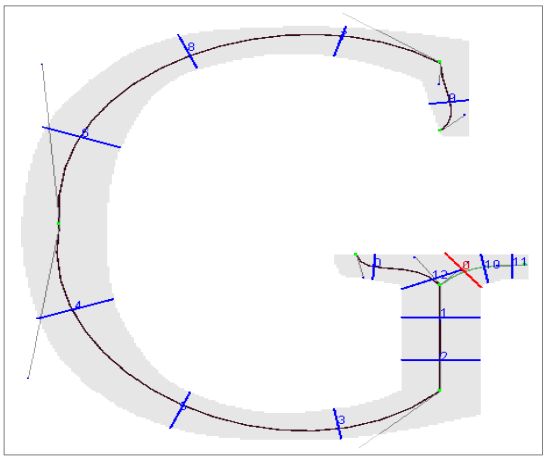


(b)

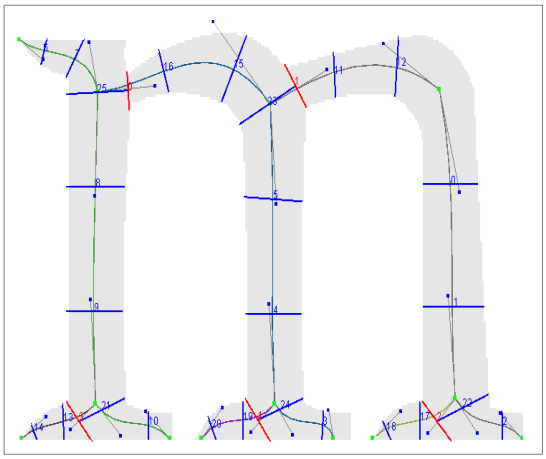
Figure 6.5: Font Liberation Serif, capital letters. Embroidery design. The red circles point out some number 2 errors (t-junctions).



(a)



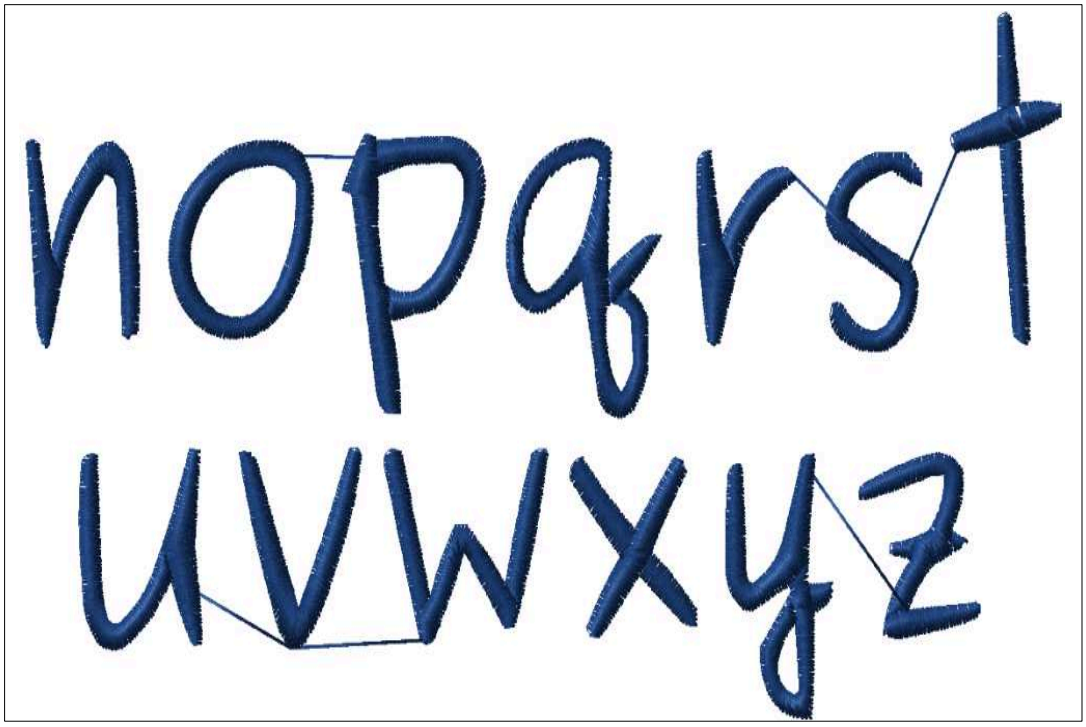
(b)



(c)

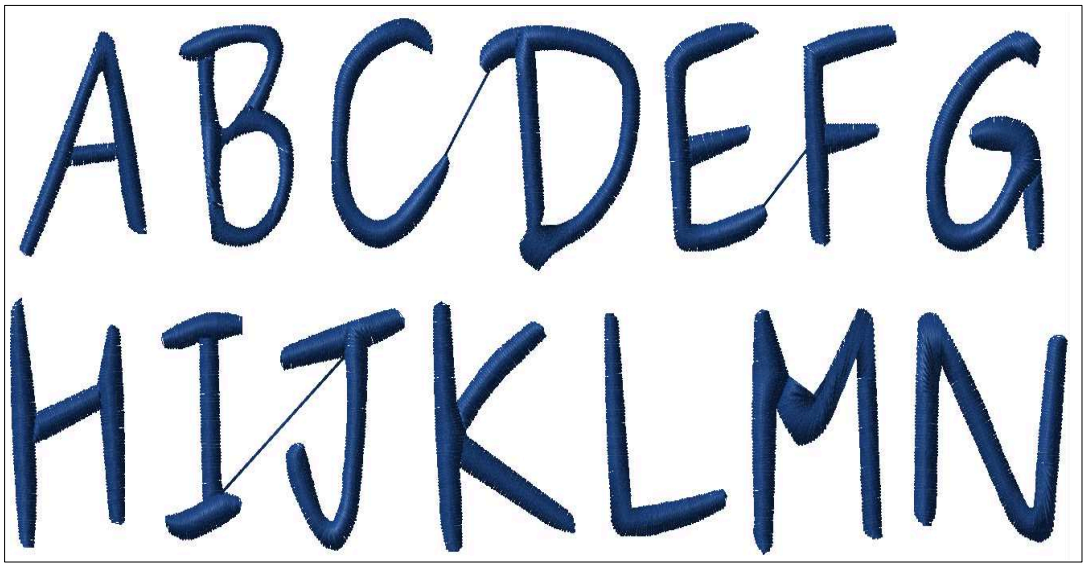
Figure 6.6: Font Liberation Serif . Segment boundaries and direction lines of a) Letter *a*. b) Letter *G*. c) Letter *m*.

(a)

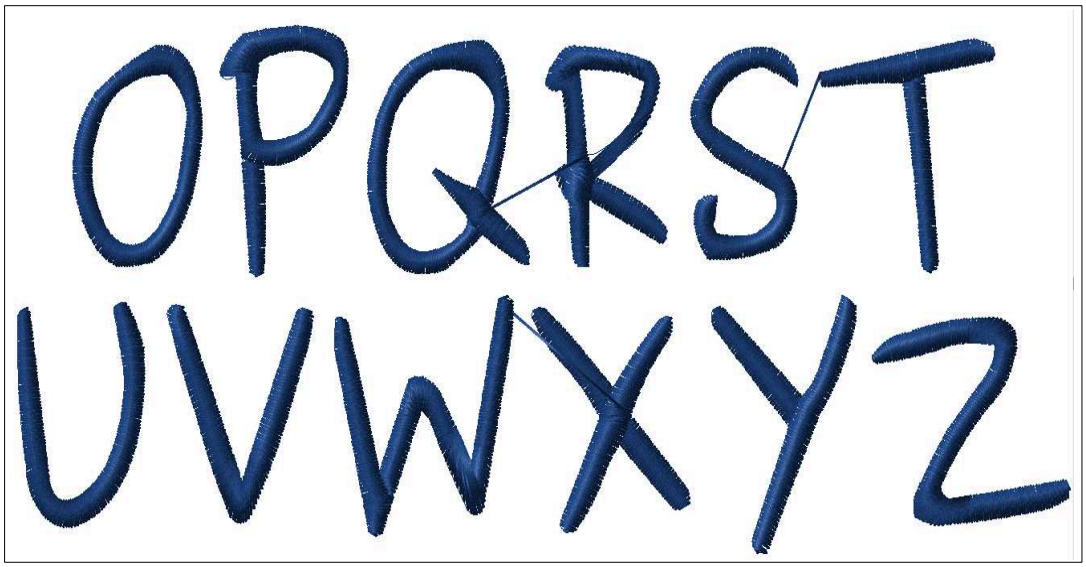


(b)

Figure 6.7: Font Purisa, lower cases. Embroidery design. The red circle in letter k in (a) shows error number 2 at a t-junction.

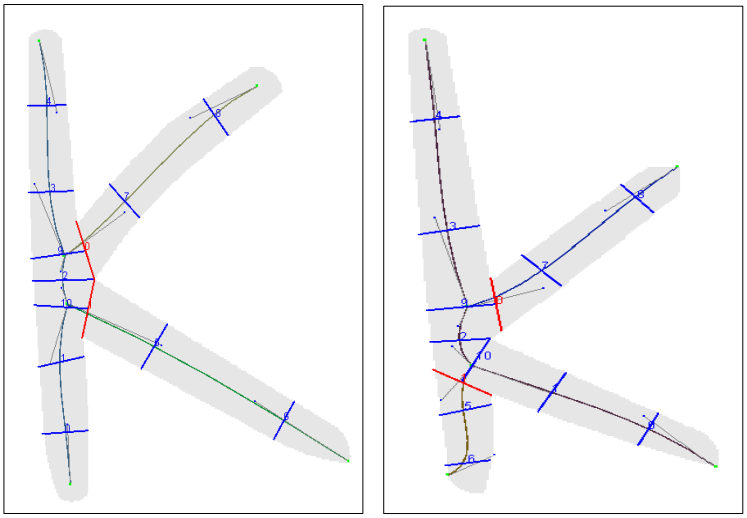


(a)

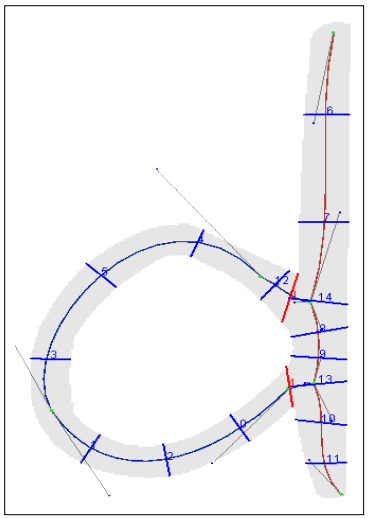


(b)

Figure 6.8: Font Purisa, capital letters. Embroidery design.



(a) (b)



(c)

Figure 6.9: Font Purisa. Segment boundaries and direction lines of a) Letter *K*. b) Letter *k*. c) Letter *d*. Note the difference between the segmentation of letter K and k.

(a)

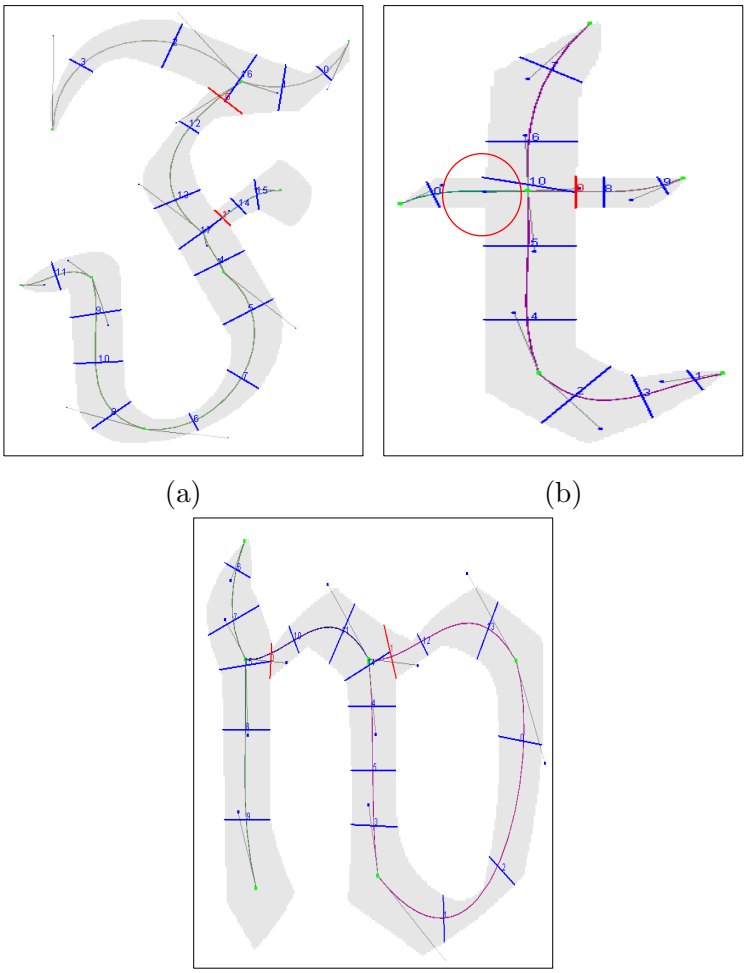
(b)

Figure 6.10: Font Eufm, lower cases. Embroidery design. The red circles in letter n and letter t in (b) show some number 4 errors where no direction lines are produced.

(a)

(b)

Figure 6.11: Font Eufm, capital letters. Embroidery design. Letter Z in (b) presents some problems during the embroidery process where the segment boundaries are not respected (error number 5).



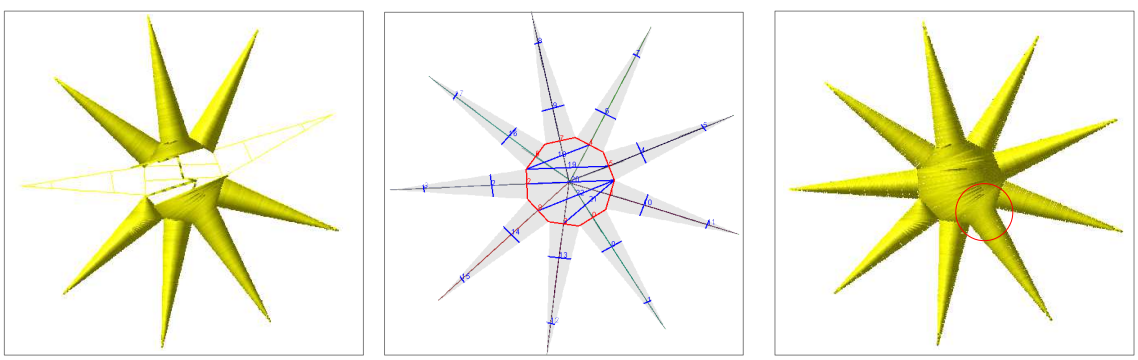
(a) (b)

(c)

Figure 6.12: Font Eufm. Segment boundaries and direction lines of a) Letter F. b) Letter t (red circle shows error number 4). c) Letter w.

6.2 Special Shapes

Another special case described in Chapter 4 was related to star-like shapes, where the segmentation obtained was not desirable for our purposes and therefore we needed to implement an extra processing of branch points and skeleton curves in order to segment all the protruding parts in this type of shapes. The segmentation after this extra processing improved results. However, despite the improved output from our software, the results in the embroidery design software still reveal some errors. The flow of the stitches does not respect some of the segment boundaries (red circles). The reason for this to occur is not clear to us, it may be that the segment boundaries intersect slightly at the end points due to a loss of precision during file i/o between the two pieces of software. These examples are shown in Figures 6.13 and 6.14. In both figures, we show the results after processing the shapes in the embroidery design software, before and after the segmentation of all the protrusions.



(a) (b) (c)

Figure 6.13: Star. a) Test in embroidery design software, result before branch point pre-processing. b) Segment boundaries and direction lines, final result. c) Test in embroidery design software, result after branch point pre-processing. Red circle shows error number 3 (segment boundaries are not respected).

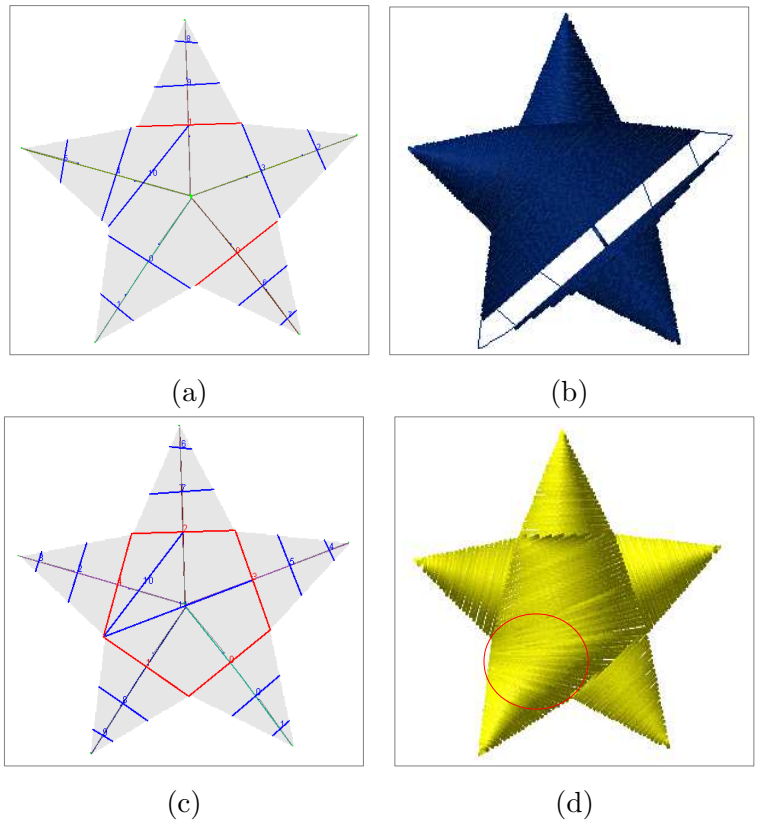
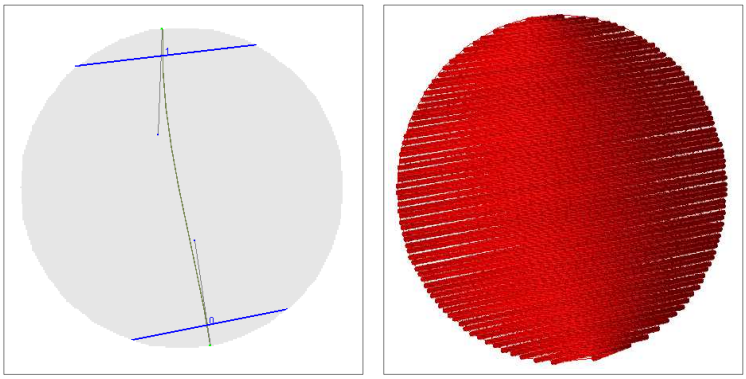


Figure 6.14: Star. a) Segment boundaries and direction lines, before branch points preprocessing. b) Test in embroidery design software. c) Segmentation, final result. d) Test in embroidery design software. Red circle shows error number 3 (segment boundaries are not respected).

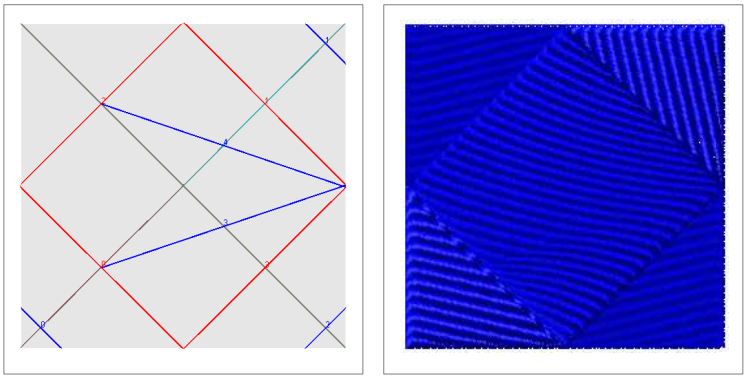
Some other shapes in this category are circles and squares. By definition the skeleton of a circle is a single centered point. However, in order to place some direction lines and produce an embroidery design, the skeleton is modified to a curve or set of curves traversing the circle. Figure 6.15 presents a circle with its skeleton, direction lines and the output from the embroidery design software.

In Figure 6.16 we show a square with it segment boundaries and direction lines. The result could be simpler. The ideal output for square shapes would be no segmentation and direction lines at the same angle as one of the sides in order to have a design uniformly embroidered.



(a) (b)

Figure 6.15: Circle. a) Segment boundaries and direction lines, final result. b) Test in embroidery design software.



(a) (b)

Figure 6.16: Square. a) Segment boundaries and direction lines, final result. b) Test in embroidery design software.

6.3 Basic Logos and Figures

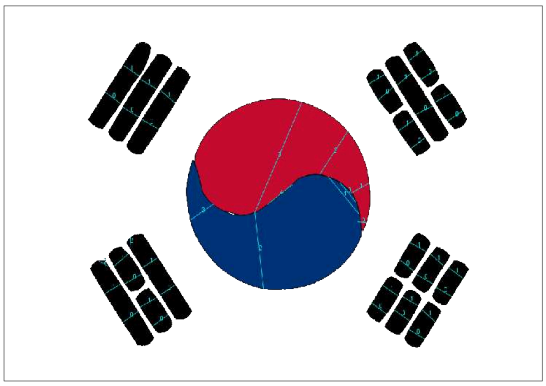
In this section we present the results for some figures which consist of several shapes and each shape was processed separately for its segmentation and placement of direction lines. Some other figures consist of a single shape but they are more complex than the lettering examples described in Section 6.1 (e.g., words).

The flag in Figure 6.17 consists of several simple shapes like rectangles. It has 20 shapes and no branch points, therefore, it does not have any segment boundary and it only required the placement of direction lines. In 6.17 b) and c) we show the output from the embroidery design software where we can see that the stitches have a continuous flow following the direction lines properly. Figure 6.18 is another example that consists of several shapes. The stitches follow the direction lines respecting the segment boundaries but as can be seen in 6.18 c) there are some gaps (blue circles). This can be solved in the embroidery software by manually adding extra direction lines or by moving the existing ones closer to the area with no stitches. Figure 6.19 b) presents the result for the embroidery design of an eye-logo and in 6.19 c) we present the 3D view of the embroidery design. The stitches have a continuous flow, however a pair of segment boundaries were placed where these line segments are not needed (blue circles). The problem can be solved by changing the pruning parameter of the skeleton from 1.2 to 1.3 as described in Chapter 4.

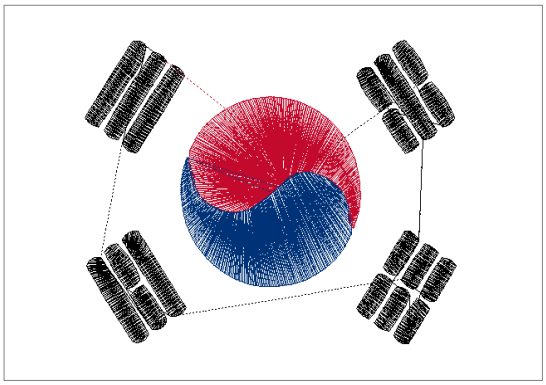
The bear in Figure 6.20 consists of a single shape and 5 branch points, the legs considered as protruding parts were segmented but the embroidery design could be improved by changing the angle of some of the direction lines, e.g., in the main part of the body.

There are some cases where the segment boundaries prevent the design from being embroidered, for example the bee's eye (blue circle) in Figure 6.21 c). This kind of problem may be solved by placing a direction line in the sharp corner of the eye to ensure its filling. Another issue with this design is that the stitches on the face do not follow the direction lines properly (blue circles). In this case, the solution could be to change the angle of the direction lines. The design consists of 19 branch points, 9 shapes and 0 overlapping maximal disks which make it part of our basic logos category.

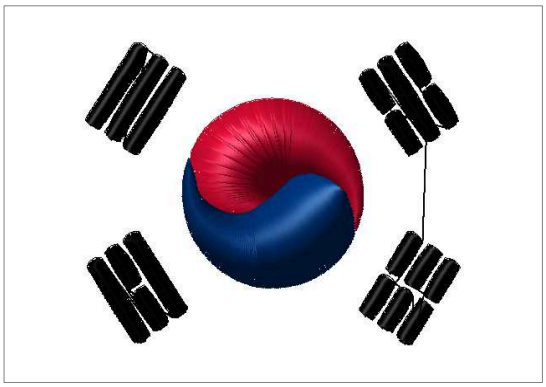
Some of the examples described in this and next section consist of several shapes as we mentioned before, and each shape is processed separately. Because of this we are using a different representation where the segment boundaries and the direction lines were drawn on the shapes. The dashed lines in some of the figures indicate the path followed by the thread for sequencing the shapes during the stitching process.



(a)

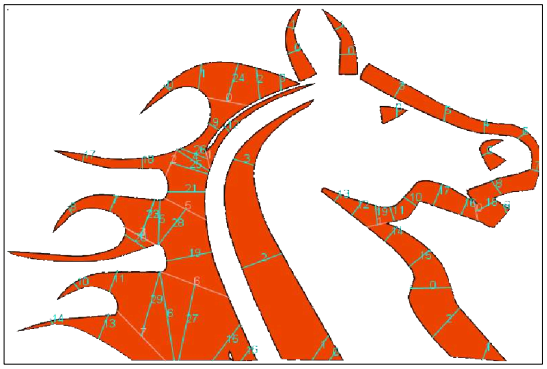


(b)

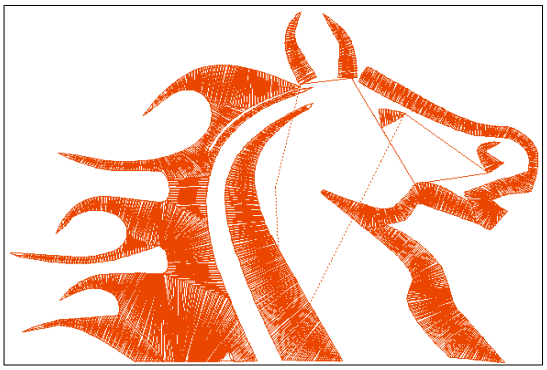


(c)

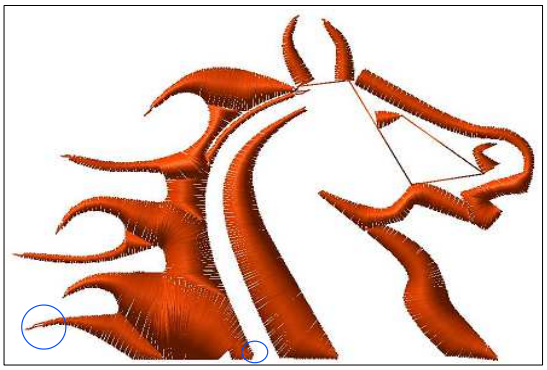
Figure 6.17: South Korea Flag. a) Direction lines. b) Test in embroidery design software. c) 3D view of embroidery design.



(a)

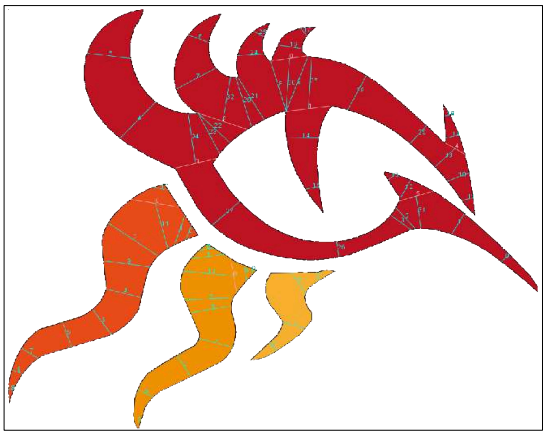


(b)



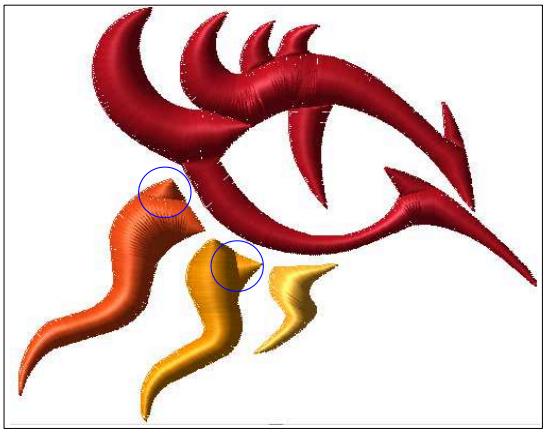
(c)

Figure 6.18: Horse. a) Segment boundaries and direction lines. b) Test in embroidery design software. c) 3D view of embroidery design. The blue circles point out some gaps. This can be solved adding a direction line or changing the position of the exiting ones (error number 6).



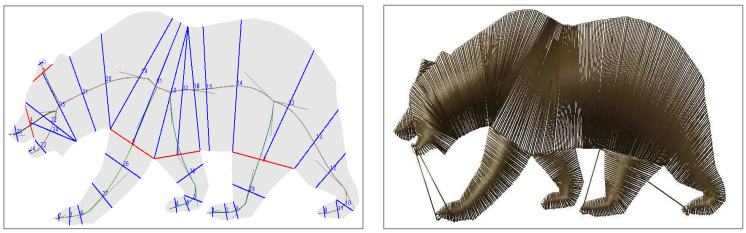
(a)

(b)



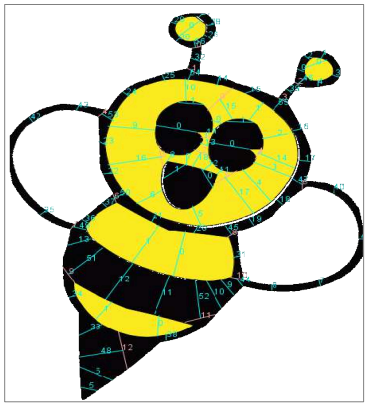
(c)

Figure 6.19: Eye logo. a) Part of the eye logo with its corresponding segment boundaries and direction lines. b) Test in embroidery design software. c) 3D view of embroidery design. Error number 2 (t-junctions) is indicated by the blue circles.

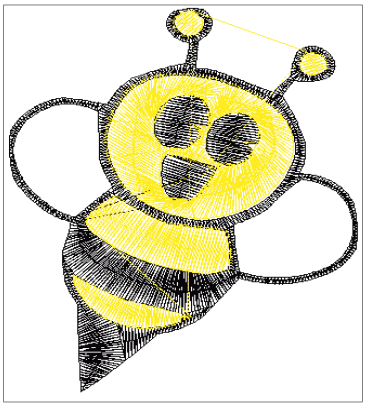


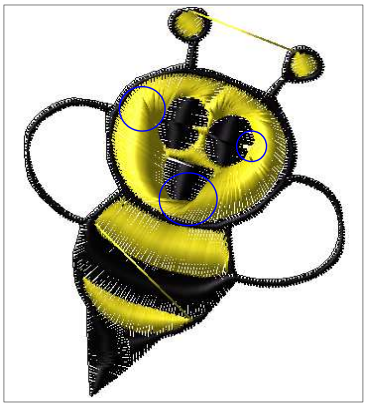
(a) (b)

Figure 6.20: Bear. a) Segment boundaries and direction lines. b) Test in embroidery design software.



(a)





(b) (c)

Figure 6.21: Bee. a) Segment boundaries and direction lines. b) Test in embroidery design software. c) 3D view of embroidery design. The blue circles indicate error number 6 (incorrect orientation of direction lines).

6.4 Complex logos

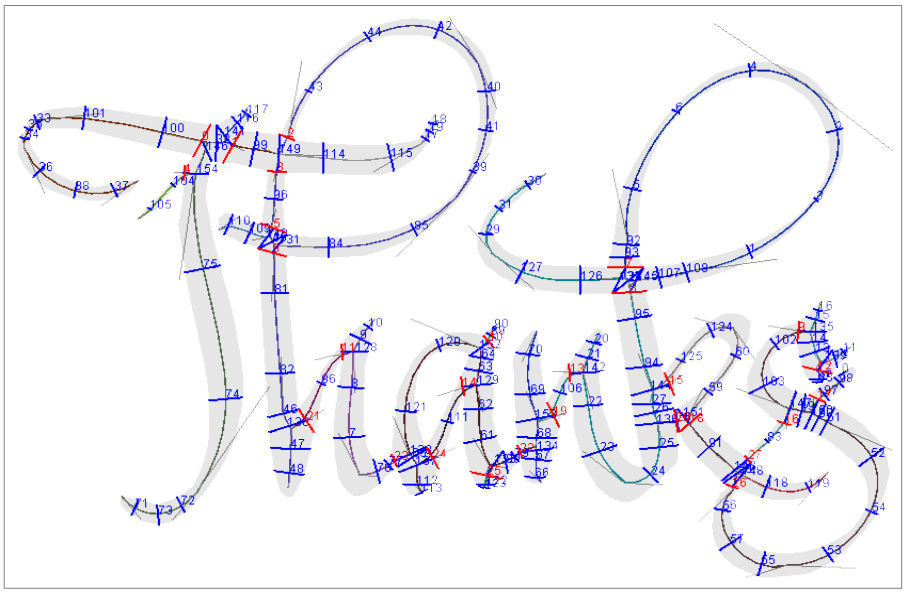
The last category in our evaluation is related to some more complex logos. This is, the examples have more than 20 branch points. Some of the figures we will present in this section are a combination of different shapes (e.g., rectangles, irregular shapes, lettering) and have several shapes inside others.

In Figure 6.22 we can see the complexity of the skeletal configuration which consists of 30 branch points. This shape was being over segmented due to overlapping maximal disks in different parts preventing the stitches from following the direction lines properly in the regions close to the branch points. In these regions there are two branch points close to each other, therefore they needed further processing. Consider the example of Figure 4.18 where we have a chart densely filled because all the branch points lie inside the maximal disk centered at each of them. For the shape shown in Figure 6.22 we use a similar chart to identify the branch points that lie inside the maximal disk of other branch points. However, in this case the chart is not completely filled, therefore only some branch points are handled as a single point (e.g., see red circles). In some other cases, it is not possible to show the skeletal representation of the whole design, then in order to show the segment boundaries and direction lines, they were drawn on the original shapes (red and blue line segments respectively).

The fish of Figure 6.23 consists of 59 shapes and 26 branch points. The stitches in this example follow the direction lines except for those in the area of the eyes (red circle). By changing the angle of the direction lines the problem would be fixed. Regarding the segmentation, we argue that the segment boundaries on the fins are not necessary (see red circles), assuming that each part of each fin consists of a single shape with no protrusions. These problem may be solved by changing the skeleton pruning parameter. Finally, the lines in the blue circles represent the stitching path.

The logo of the University of Ottawa in Figure 6.24 is a combination of lettering with some other irregular shapes and consists of 12 shapes and 23 branch points. The segmentation and direction lines are shown in Figure 6.24 a) while Figure in 6.24 b) we present the output from the embroidery design software where it can be seen that the stitches have a continuous flow respecting the segment boundaries. Finally we present part of a brain in Figure 6.25 where it is possible to see the complexity of the skeleton. This is one of the most complex examples we have tested it has 56 shapes and 103 branch points. As we can see there is a great number of segment boundaries and direction lines and part of the shape is being over-segmented causing gaps in the embroidery

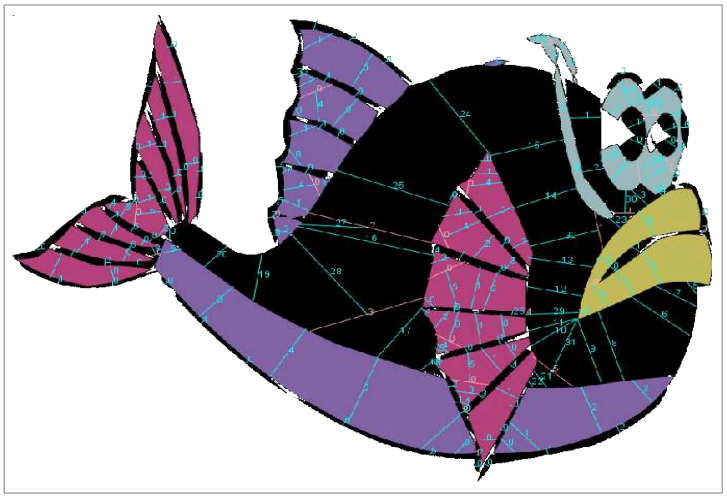
design as shown in Figure 6.26, therefore, the branch points in this example also need a further processing in order to obtain more optimal results. In this case we are not handling the multiple branch points in the different regions as a single branch point as explained in Chapter 4. Despite these issues with the segmentation, most of the shape was properly processed by the design software, by this we mean that the embroidery follows the direction lines. There are several regions where branch points are very close to each other causing an over segmentation (e.g., in the 3 or 4-way junctions, see black circles) and need further processing.



(a)

(b)

Figure 6.22: Thanks. a) Segment boundaries and direction lines. b) Embroidery design. The red circles indicate some number 6 errors (incorrect orientation of direction lines).

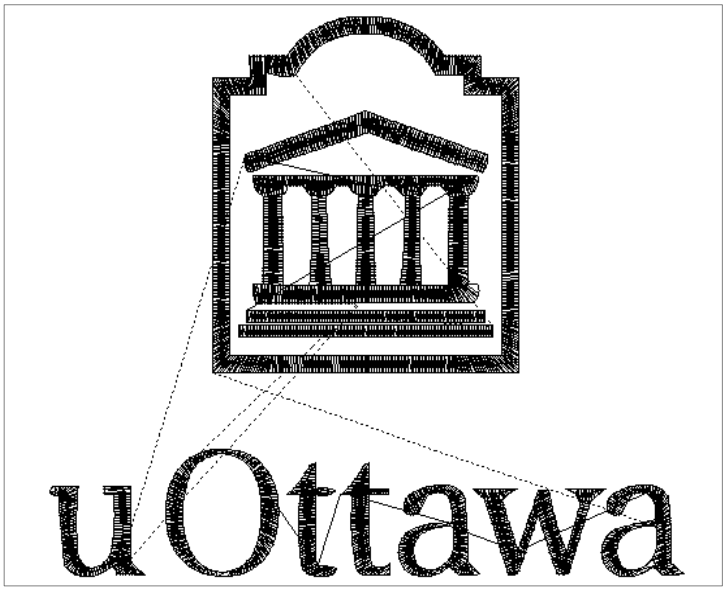


(a)

(b)

Figure 6.23: Fish. a) Segment boundaries and direction lines. b) 3D view of embroidery design. The red circles on the fins and on the eyes indicate some drawbacks in the placement of segment boundaries (error number 2) and direction lines (error number 5) respectively. The blue circles show the path followed by the thread during the embroidery process.

(a)



(b)

Figure 6.24: University of Ottawa logo. a) Segment boundaries and direction lines for combination of different types of shapes, including lettering. b) Embroidery design.

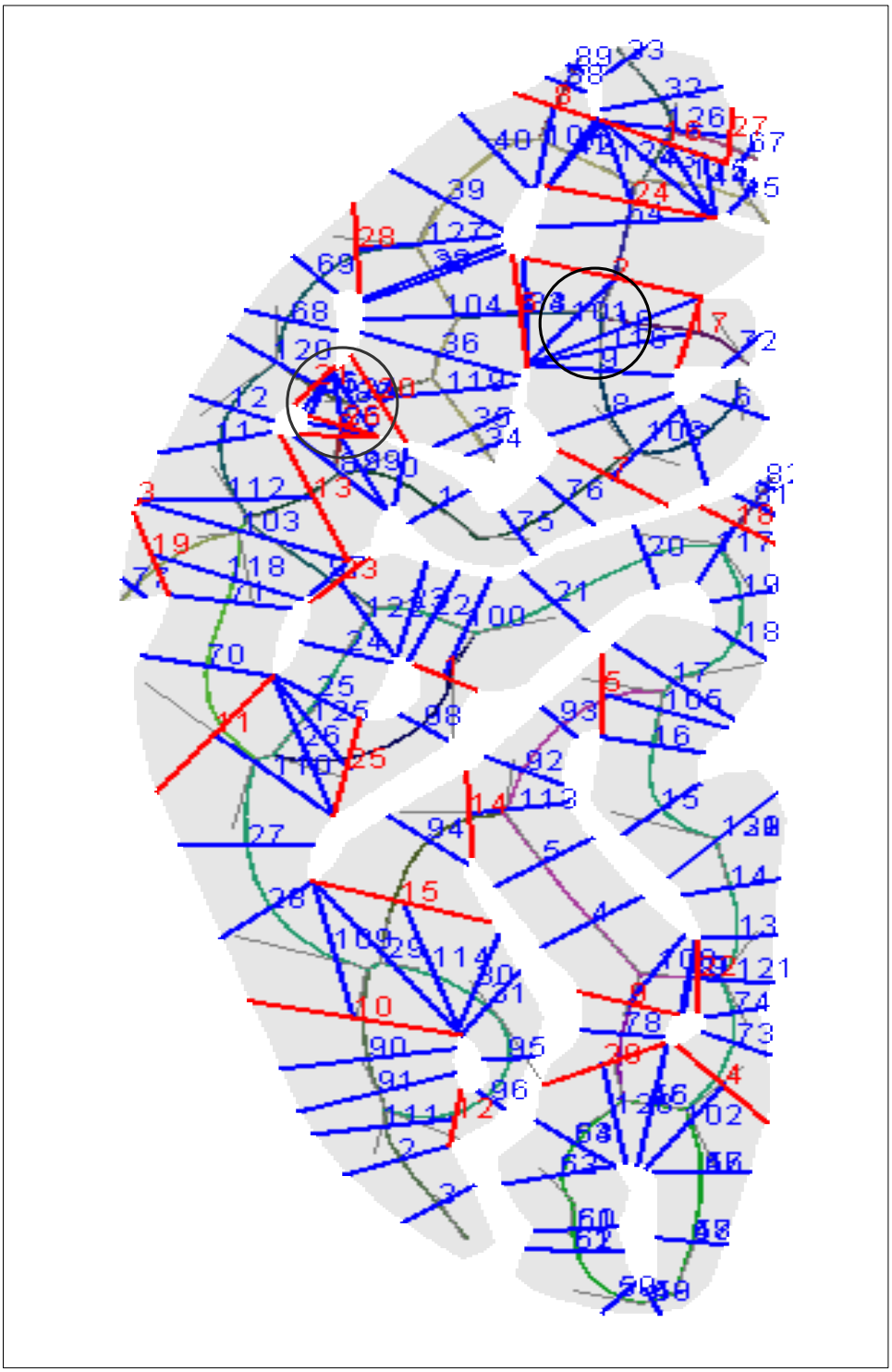


Figure 6.25: Skeleton of Yves Guillou brain segment. This figure shows an example of a complex skeleton. The black circles indicate some number 5 errors: overlapping multi-way junctions.

Figure 6.26: Yves Guillou brain. Output from embroidery design software.

6.5 Summary

In this chapter we have presented some results obtained from our implementation. We tested different types of shapes which were classified into four categories: lettering, special shapes, basic logos and complex logos. The special shapes included shapes like stars, circles and squares. Other shapes were classified as basic or complex logos according to their complexity.

The results obtained for the lettering examples were satisfactory in most cases. For the special shapes, even when the results were improved after the branch point preprocessing, they are not completely favorable. This could be due to a slight intersection of segment boundaries. Finally, the results obtained for the basic and complex logos are often successful. In some of these examples are gaps mainly due to poor placement of direction lines. Probably some more direction lines need to be placed or the orientation of direction lines needs to be changed in order to produce better results. Other cases presented an over segmentation due to the complexity of the skeleton and multiple branch points close to each other in different regions of the shape. This scenario generates gaps or that some stitches overlap. Despite these drawbacks, we argue that our results respect the shape segmentation and the direction lines are followed properly making them acceptable for most cases.

Chapter 7

Conclusions and Future Work

The goal of our research was the analysis of 2D shapes and the development of techniques for shape segmentation in the automation of embroidery designs. For this purpose, we also devised some techniques for the placement of direction lines that control the orientation of stitches in a embroidery design. In this chapter we summarize our work, we describe the limitations of the proposed method and give future direction of our research.

7.1 Summary

The creation of embroidery designs requires a shape to be divided into parts, hence one of the goals of our work was to develop a technique for shape segmentation based on the shape skeleton. Depending on the desired design, one needs to select a stitching style. In our work we were interested in satin-stitches which need to be given an orientation to make them flow smoothly through the shape in a quality embroidery design.

We reviewed literature related to different existing methods for shape skeleton computation including grassfire simulation, Voronoi diagrams, thinning methods and methods based on the distance transform.

We based our approach on the shape skeleton representation implemented by Solis Montero and Lang [43] which in turn is based on the feature transform. Therefore, we provided a detailed description of this method. Using the information from the skeleton and the feature transform, we developed a technique for shape segmentation. Our approach considers the skeleton branch points, the curves related to them and their closest points on the boundary to establish the criteria for the placement of segment boundaries

and obtain a segmented shape. During shape segmentation we faced some issues regarding the skeleton configuration of shapes with squared ends. These configurations led to extraneous segment boundaries which were undesirable for the embroidery design. This was solved by removing some of the skeleton branches. Another skeleton configuration for which we devised special processing is found in shapes like stars and squares. The skeleton of this type of shapes has all its branch points close to each other preventing a desirable segmentation for our purposes. The solution for these cases was to treat all the branch points as a single point, ignoring the curves between branch points and placing segment boundaries on the remaining curves.

The second part of our work used the regions obtained from the segmentation step, the skeleton curves and the feature transform to find points on the boundary that allowed us to find pairs of points to define the direction lines and their position on the skeleton to control the flow of the stitches in an embroidery design. The number of direction lines placed on each skeleton curve was based on the length of the curve taking into account if the curve was intersected by a segment boundary or not. In the placement of direction lines we also found some special cases, e.g., when a shape has sharp corners, the skeleton is split into two curves and we needed to place a direction line at the join of the skeleton curves. Another example, consists of small shapes with skeleton represented by a single point. In these cases we computed a middle point using the shape size and used it for the computation and placement of a direction line. After obtaining both, segment boundaries and direction lines we computed their intersection with the shape boundary curves in order to define the final segmentation and direction lines.

Finally, we showed several examples of the results obtained from our implementation. We divided the examples into different categories according to their complexity. These categories are: lettering, special shapes like stars, circles and squares, and basic and complex logos. We also tested our results in a commercial software specialized in the design of embroidery patters to visualize how the final output would look like. In most cases we obtained satisfactory results however, there is still room for improvement mainly in the segmentation of shapes with multiple branch points close to each other.

7.2 Limitations

From the results described in the last chapter, we can see some of the limitations of our implementation. One of them occurs during the removal of skeleton branches from shapes with squared ends, the results were improved but only for those shapes where

the branches had the same length or one of them was slightly larger than the other. The cases where the branches were not removed can be solved by changing the skeleton pruning parameter as described in previous chapters. We also have scenarios of shapes with multiple branch points. For simple lettering examples, the results were improved by removing some segmenting lines after the branch points pre-processing. However, in the case of star shapes, even when the results were improved, we detected some issues after testing them in the embroidery design software. This could be due to an overlap of the endpoints of the segmenting line segments. In the case of squares, the shapes are completely embroidered but we are over-segmenting the shape.

All the shapes described above have all their branch points close to each other. If we compute the maximal disk centered at each of them, the other branch points are contained inside the disk. But there are cases where a shape has many branch points and only some of them are in the same region. In these cases we are obtaining an over segmented shape, such cases can be seen in Figure 6.25, for instance.

Another drawback, is related to the placement of direction lines. Some of the results show some gaps in the output obtained from the embroidery design software. In these cases a possible lack of direction lines is causing the stitches not to reach some regions of the shape. There are some other shapes where the results are desirable but they can be improved by changing the orientation of some direction lines. The manual user intervention was reduced considerably in most cases, therefore the angle of the direction lines can be done directly in the design software without being a difficult or time consuming task. This scenario can be seen in the bear of Figure 6.20.

Regarding the processing time, it is acceptable according to the different shapes and their complexity, but there are some algorithms used for the computation of the intersections of the lines with the boundary curves that could be modified in order to improve their performance. In this case, for each segment line endpoint we are looping over all the curves until we find the intersected one. The search can be improved by using an acceleration data structure, e.g., a kd-tree, in order to reduce the time of this step.

7.3 Future work

For future work, we are interested in the enhancement of our implementation, in particular, we would like to address the issues related to the shape over segmentation. As we described, we dealt with some cases where all the branch points of a shape were very

close to each other defining undesirable segment boundaries but now we need to deal with cases where just some branch points share the same aspect ratio. This situation can be seen in several parts of the same shape, therefore, we need to identify which branch points should be grouped together and decide the way to process them.

Another improvement to our implementation is related to the shape skeleton. Currently, the continuity of curves that belong or share a branch point is taken into account for the placement of segment boundaries but sometimes this criterion does not produce desirable results (e.g., shapes where the segmentation at T-junctions follow only local continuity), then, some modifications could be done to the skeleton in order to fix some "irregularities" due to the shape protrusions and change the continuity of the curves improving the segmentation. We would also like to study if our segmentation results correspond to the way the human perception decomposes shapes.

Also, we would like to improve the placement of direction lines by increasing the number of lines in each shape region or/and by changing the orientation of some lines in order to make the flow of the stitches smoother obtaining more pleasant results and eliminate the need for user's manual intervention. The main goal is to be able to handle as many cases as possible.

Bibliography

[1] T. Adamek and N. O'Connor. Efficient Contour-Based Shape Representation and Matching. In *Proceedings of 5th ACM SIGMM International Workshop on Multimedia Information Retrieval*, pages 138–143. ACM, 2003.

[2] O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gartner. A Novel Type of Skeleton for Polygons. *Journal of Universal Computer Science*, 1(12):752–761, 1995.

[3] C. Arcelli. Pattern Thinning by Contour Tracing. *Computer Graphics and Image Processing*, 17(2):130–144, 1981.

[4] D. Attali, J.-D. Boissonnat, and H. Edelsbrunner. Stability and Computation of Medial Axes-a State-of-the-Art Report. In *Mathematical Foundations of Scientific Visualization, Computer Graphics and Massive Data Exploration*, pages 109–125. Springer, 2009.

[5] X. Bai, X. Yang, D. Yu, and L.J. Latecki. Skeleton-Based Shape Classification Using Path Similarity. *Journal of Pattern Recognition and Artificial Intelligence*, 22(4):733–746, 2008.

[6] H. Blum. A Transformation for Extracting New Descriptors of Shape. In *Models for Perception of Speech and Visual Form*, pages 362–380. Wathen-Dunn W., ed, 1967.

[7] H. Blum. Biological Shape and Visual Science. *Journal of Theoretical Biology*, 38(2):205–287, 1973.

[8] W. Brandt and V.R. Algazi. Continuous Skeleton Computation by Voronoi Diagram. *CVGIP: Image Understanding*, 55(3):329–338, 1992.

[9] D. Breen, W. Regli, and M. Peysakhov. Computer Graphics I, Curve Drawing Algorithms. `https://www.cs.drexel.edu/`. [Online; accessed September-2012].

[10] J. Chaussard, M. Couprie, and H. Talbot. Robust Skeletonization Using the Discrete Lambda-Medial Axis. *Pattern Recognition Letters*, 32(9):1384–1394, 2011.

[11] F. Chazal and A. Lieutier. The Lambda-medial Axis. *Graphical Models*, 67(4):304–331, 2005.

[12] X. Chen. Embroidery Modelling and Rendering. Master's thesis, University of Waterloo, 2011.

[13] L. Da Fontoura Costa and R. Marcondes Cesar. *Shape Analysis and Classification: Theory and Practice*, chapter 1. CRC Press, 2009.

[14] M. Demuth, F. Aurenhammer, and A. Pinz. Straight Skeleton for Binary Shapes. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops(CVPRW)*, pages 9–16, 2010.

[15] L.G. Domakhina. Skeleton-Based Segmentation and Decomposition of Raster Pair of Shapes. *Pattern Recognition and Image Analysis*, 20(3):293–302, 2010.

[16] L. De Floriani and M. Spagnuolo. *Shape Analysis and Structuring*, chapter 5. Springer, Berlin, 2008.

[17] D. Fussell and K.R. Subramanian. Fast Ray Tracing Using K-D Trees. Technical report, University of Texas at Austin, Austin, TX, USA, 1988.

[18] J. Giesen, B. Miklos, M. Pauly, and C. Wormser. The Scale Axis Transform. In *Proceedings of the 25th Annual Symposium on Computational Geometry*, pages 106–115. ACM, 2009.

[19] A.S. Glassner. *Graphic Gems*, chapter 8. Academic Press, INC., 1990.

[20] A.S. Glassner. *Graphic Gems*, chapter 11. Academic Press, INC., 1990.

[21] W.-B. Goh. Strategies for Shape Matching Using Skeletons. *Computer Vision and Image Understanding*, 110(3):326–345, 2008.

[22] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A Non-Photoralistic Lighting Model for Automatic Technical Illustration. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 447–452. ACM, 2004.

[23] A. Gooch, J. Long, L.J., and A. Estey. Viewing Progress in Non-Photorealistic Rendering through Heinlein's Lens. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 165–171. ACM, 2010.

[24] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*, chapter 1. AK Peters Ltd, 2001.

[25] P. Haeberli. Painting by Numbers. *SIGGRAPH Computer Graphics*, 24(4):207–214, 1990.

[26] A. Hertzmann. Painterly Rendering with Curve Brush Strokes of Multiple Sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 453–460. ACM, 1998.

[27] A. Hertzmann and D. Zorin. Illustrating Smooth Surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 517–527. ACM Press/Addison-Wesley Publishing Co., 2000.

[28] W.H. Hesselink and J.B.T.M. Roerdink. Euclidean Skeletons of Digital Image and Volume Data in Linear Time by the Integer Medial Axis Transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(12):2204–2217, 2008.

[29] H. Kang, S. Lee, and C.K. Chui. Coherent Line Drawing. In *Proceedings of the 5th International Symposium on Non-Photorealistic Animation and Rendering*, pages 43–50. ACM, 2007.

[30] H. Kang, S. Lee, and C.K. Chui. Flow-Based Image Abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):62–76, 2009.

[31] J.E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg. State of 'Art': A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*, (99):1–20, 2012.

[32] L. Lam, S.-W. Lee, and C.Y. Suen. Thinning Methodologies-A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992.

[33] F. Laymarie and M.D. Levine. Simulating the Grassfire Transform using an Active Contour Model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):56–75, 1992.

[34] H. Lee, S. Seo, and K. Yoon. Directional Texture Transfer with Edge Enhancement. *Computer Graphics*, 35(1):81–91, 2011.

[35] D. Macrini, S. Dickinson, D. Fleet, and K. Siddiqi. Bone Graphs: Medial Shape Parsing and Abstraction. *Computer Vision and Image Understanding*, 115(7):1044–1061, 2011.

[36] G. Malandain and S. Fernández-Vidal. Euclidean Skeletons. *Image and Vision Computing*, 16(5):317–327, 1998.

[37] N. Mayya and V.T. Rajan. An Efficient Shape Representation Scheme Using Voronoi Skeletons. *Pattern Recognition Letters*, 16(2):147–160, 1995.

[38] N. Mayya and V.T. Rajan. Voronoi Diagrams of Polygons: A Framework for Shape Representation. *Journal of Mathematical Imaging and Vision*, 6(4):355–378, 1996.

[39] A. Meijster, J.B.T.M. Roerdink, and W.H. Hesselink. A General Algorithm for Computing Distance Transforms in Linear Time. *Mathematical Morphology and Its Applications to Image and Signal Processing*, 18:331–340, 2002.

[40] Merriam-Webster. Satin-Stitch. "`http://www.merriam-webster.com/dictionary/`. [Online; accessed July-2012].

[41] X. Mi and D. DeCarlo. Separating Parts from 2D Shapes using Relatability. In *IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[42] B. Miklos, J. Giesem, and M. Pauly. Discrete Scale Axis Representation of 3D Geometry. *ACM Transactions on Graphics*, 29(4):101:1–101:10, 2010.

[43] A. Solis Montero and J. Lang. Skeletal Pruning by Contour Approximation and the Integer Medial Axis. *Computers & Graphics*, 36(5):477–487, 2012.

[44] M. Naouai, A. Hammouda, S. Jalel, and C. Weber. NURBS Skeleton: A New Shape Representation Scheme using Skeletonization and NURBS Curves Modeling. In *16th Iberoamerican Congress on Progress in Pattern Recognition, Image Analysis, Computer Vision and Applications*, pages 197–205. Springer-Verlag Berlin,Heidelberg, 2011.

[45] C.W. Niblack, P.B. Gibbons, and D.W. Capson. Generating Skeleton and Centerlines from Distance Transform. *CVGIP Graphical Models and Image Processing*, 54(5):420–437, 1992.

[46] P. O'Donovan and D. Mould. Felt-Based Rendering. In *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering*, pages 55–62. ACM, 2006.

[47] R.L. Ogniewicz and O.Kubler. Hierarchic Voronoi Skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[48] T. Pavlidis. A Thinning Algorithm for Discrete Binary Images. *Graphics and Image Processing*, 13(2):142–157, 1980.

[49] G. Pomaska. Between photo-realism and non-photorealistic rednering - modeling urban areas for real time vr. In *International Workshop on Vision Techniques Applied to the Rehabilitation of City Centres*, 2004.

[50] F.B. Prince, J.-H. Chern, and Carnegie Mellon University Engineering Design Research Center. Geometric Abstractions using Medial Axis Transformation, August 2012.

[51] W.R. Quadros, B.Gurumoorthy, K.Ramaswami, and F.B. Prinz. Skeletons for Representation and Reasoning in Engineering Applications. *Engineering with Computers*, 17(2):186–198, 2001.

[52] U. Ramer. An Iterative Procedure for the Polygonal Approximation of Plane Curves. *Computer Graphics Image Processing*, 1(3):244–256, 1972.

[53] D. Reniers and A. Telea. Skeleton-Based Hierarchical Shape Segmentation. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*, pages 179–188. IEEE Computer Society, 2007.

[54] H. Rom and G. Medioni. Hierarchical Decomposition and Axial Shape Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):973–981, 1993.

[55] M.P. Salisbury, S. Anderson, R. Barzel, and D.H. Salesin. Interactive Pen-and-Ink Illustration. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 101–108. ACM, 1994.

[56] M.P. Salisbury, M.T. Wong, J.F. Hughes, and D.H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 401–406. ACM, 1997.

[57] J. Schlecht and B. Ommer. Contour-Based Object Detection. In *Proceedings of the British Machine Vision Conference*. BVA Press, 2011.

[58] J. Shah. Gray Skeletons and Segmentation of Shapes. *Computer Vision and Image Understanding*, 99(1):96–109, 2005.

[59] J. Shah. Skeletons and Segmentation of Shapes. Technical report, Department of Mathematics, Northeastern University, 2005.

[60] K. Siddiqi and S.M. Pizer. *Medial Representations: Mathematics, Algorithms and Applications*, chapter 1. Springer, 2008.

[61] A. Torsello and E. R. Hancock. A Skeletal Measure of 2D Shape Similarity. *Computer Vision and Image Understanding*, 95(1):1–29, 2004.

[62] P.J. Vermeer. *Medial Axis Transform to Boundary Representation Conversion*. PhD thesis, Pardue University, 1994.

[63] J. Vince. *Mathematics for Computer Graphics*, chapter 10. Springer., 2010.

[64] G. Winkenbach and D.H. Salesin. Computer-Generated Pen-and-Ink Illustrations. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 91–100. ACM, 1994.

[65] G. Winkenbach and D.H. Salesin. Rendering Parametric Surfaces in Pen-and-Ink. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 469–476. ACM, 1996.

[66] Y. Xia. Skeletonization Via the Realization of the Fire Front's Propagation and Extraction in Digital Binary Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(10):1076–1086, 1989.

[67] J.T. Zeng, R. Lakaemper, X.W. Yang, and X. Li. 2D Shape Decomposition Based on Combined Skeleton-Boundary. *Lecture Notes in Computer Science*, 5359(2):682–691, 2008.

[68] D. Zhang and M. Chen Yi Lim. An Efficient and Robust Technique for Region-Based Shape Representation and Retrieval. In *Proceedings of 6th IEEE/ACIS International Conference on Computer and Information Science*, pages 801–806, 2007.

[69] D. Zhang and G. Lu. A Comparative Study of Curvature Scale and Fourier Descriptors for Shape-Based Image Retrieval. *Journal of Visual Communication and Image Representation*, 14(1):39–57, 2003.

[70] D. Zhang and G. Lu. Review of Shape Representation and Description Techniques. *Pattern Recognition*, 37(1):1–19, 2004.