An Efficient Font Design Method

by

Murtaza Ali Khan

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

January, 2001

An Efficient Font Design Method

BY

MURTAZA ALI KHAN

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

KING FAHD UNIVERSITY
OF PETROLEUM & MINERALS
Dhahran, Saudi Arabia

JANUARY 2001

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Murtaza Ali Khan**

under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE.

Thesis Committee

Dr. Muhammad Sarfraz (Chairman)

Dr. Jarallah S. AlGhamdi (Member)

Dr. Kalim Uddin Qureshi (Member)

Department Chairman

Dean, College of Graduate Studies

14-1-2001

Date

عمادة الدراسات العليا

DEANSHIP OF GRADUATE STUDIES

Dedicated to

All those

who spent some time

in KFUPM

as

Research Assistant(RA)

ACKNOWLEDGEMENTS

In the name of Allah,Most Gracious, Most Merciful

All praise and glory to Alimgithy Allah (SWT) who gave me courage and patience to carry out this work. Peace and blessing of Allah be upon last Prophet Muhammad (PBUH).

Acknowledgement is due to King Fahd University of Petroleum and Minerals for providing support for this work.

My deep appreciation goes to my thesis advisor Dr. Muhammad Sarfraz for his constant help, guidance and the countless hours of attention he devoted throughout the course of this research work. His priceless suggestions made this work interesting and learning for me.

Thanks are due to my thesis committee members Dr. Jarallah Al-Ghamdi and Dr. Kalim Uddin Qureshi for their interest, invaluable cooperation and support.

Acknowledgement is due to M. Najum-ul- Haque, who initiated work on this project.

Special thanks are due to my colleagues and friends for their help and encouragement. A few of them are Saqib, Zeeshan, Arshad, Owais and Alivi. They have made my work and stay at KFUPM very pleasant.

And finally, my heartfelt thanks to my parents and other family members, their prayers and support and are always with me.

Contents

List of Tables

List of Figures

THESIS ABSTRACT

Name: Murtaza Ali Khan

Title: An Efficient Font Design Method

Degree: MASTER OF SCIENCE

Major Field: Computer Science

Date of Degree: January 2001

Two fundamental techniques of fonts representation are bitmap and outline. The outline fonts are produced from a gray-level image obtained by scanning the original characters drawn on paper. Contour points are obtained from the gray-level image and significant points are determined from contour points. Splines are approximated/interpolated to these significant points.

In traditional font design systems, the user specifies significant points by some interactive device (e.g. mouse). The system then generates a curve whose shape depends on specified points and mathematical model of the curve.There are several drawbacks of this method. The user has to be familiar with the underlying mathematical model of the system in order to use it properly. Human intervention and changing set of points many times makes this method slow. Accuracy up to desired tolerance limit is difficult to achieve.

We proposed an efficient font design method that automate the process of font design by finding minimal number of significant points ,removing noise from contour, fitting cubic Bezier to significant points and Optimizing the closeness of fit between original digitized curve and our parametric curve.

King Fahd University of Petroleum and Minerals, Dhahran.
January 2001

خلاصة الرسالة

اسم الطالب: مرتضى علي خان
عنوان الرسالة: طريقة فعّالة لتصميم الخطوط
الدرجة: ماجستير في العلوم
التخصص: علوم الحاسب الآلي
تاريخ الشهادة: يناير 2001م

إن الأسلوبين الأساسيين في تمثيل الخطوط هما Bitmap و Outline. يمكن الحصول على الخطوط الكفافية Outline Font للصورة الرمادية عن طريق المسح الألكتروني للحروف المرسومة على الورق، حيث يتم إيجاد النقاط الكنتورية للصورة الرمادية و تحديد النقاط الهامة من هذه النقاط الكنتورية و من ثم إيجاد منحنى يمر عبر هذه النقاط.
في انظمة تصميم الخطوط التقليدية Traditional Font Design Systems يحدد المستخدم النقاط الهامة Significant Points بإستخدام أجهزة خاصة (كالفارة مثلا) و من ثم يقوم النظام بتوليد منحنى يعتمد على هذة النقاط الهامة و نموذج رياضي لهذا المنحنى. يعيب هذا الأسلوب أن المستخدم يجب أن يكون على دراية بالنموذج الرياضي للنظام حتى يتم إستخدامه بالشكل الصحيح. كما أن التدخل البشري وتكرار تغيير النقاط يجعل هذا الأسلوب بطيئا, إضافة إلى صعوبة تحقيق مستوى أدنى و مطلوب من الدقة.
في هذه الرسالة سنطرح طريقة كفؤة لتصميم الخطوط تجعل عملية تصميم الخطوط أوتوماتيكيه و ذلك عن طريق إيجاد عدد أدنى من النقاط الهامة , إزالة الشوائب من الكنتور , مطابقة المنحنى البيزييري التكعيبي Cubic Bezier Curve للنقاط الهامة و تحسين دقة مطابقة المنحنى بين النقاط الأصلية و المنحنى الذي نريد إيجاده.

جامعة الملك فهد للبترول و المعادن
الظهران، المملكة العربية السعودية
يناير 2001م

Chapter 1

INTRODUCTION

There are number of applications where finding a mathematical curve description of the desired shape is benefical e.g. design of automobiles, ships, aircrafts, equipments and of course fonts.

Two fundamental techniques of fonts representation are bitmap and outline. §1.4.3 discuss what are the drawbacks of bitmap fonts. Most of the contemporary desktop publishing systems use outline fonts. This study is related to outline representation of fonts.

1.1 Problem

The outline fonts are produced from a gray-level image obtained by scanning the original characters drawn on paper. Contour points are obtained from the gray-

level image and significant points are determined from contour points. Splines are approximated/interpolated to these significant points.

In traditional font design systems, the user specifies significant points by some interactive device (e.g. mouse). The system then generates a curve whose shape depends on specified points and mathematical model of the curve. If the obtained shape is not accurate or desirable then designer has to modify or specify new set of points.

There are several drawbacks of this method. The user has to be familiar with the underlying mathematical model of the system in order to use it properly. Human intervention and changing set of points many times makes this method slow. Accuracy up to desired tolerance limit is difficult to achieve.

1.2 Solution

The solution of above problem is to automate the process of

- Extraction of contour or boundary points.
- Determination of significant/control points.
- Fitting of parametric curve to approximate the digitized curve up to desired tolerance limit.

1.3 Objectives

This study aims to proposed an efficient method of outline font design with following features:

1. Finding minimal number of significant points from the contour points.
2. Removing any noise from contour, if present.
3. Fitting suitable spline to these significant points.
4. Optimizing the closeness of fit between original digitized contour and our parametric curve.
5. Automation of all above steps.

1.4 Representation of Fonts

There are two fundamental techniques for storing fonts in computer.

1. Bitmap
2. Outline

1.4.1 Bitmap

A bitmap is characterized by the width and height of the image in pixels. Number of bits per pixel determines number of shades of gray or colors that can be represented.

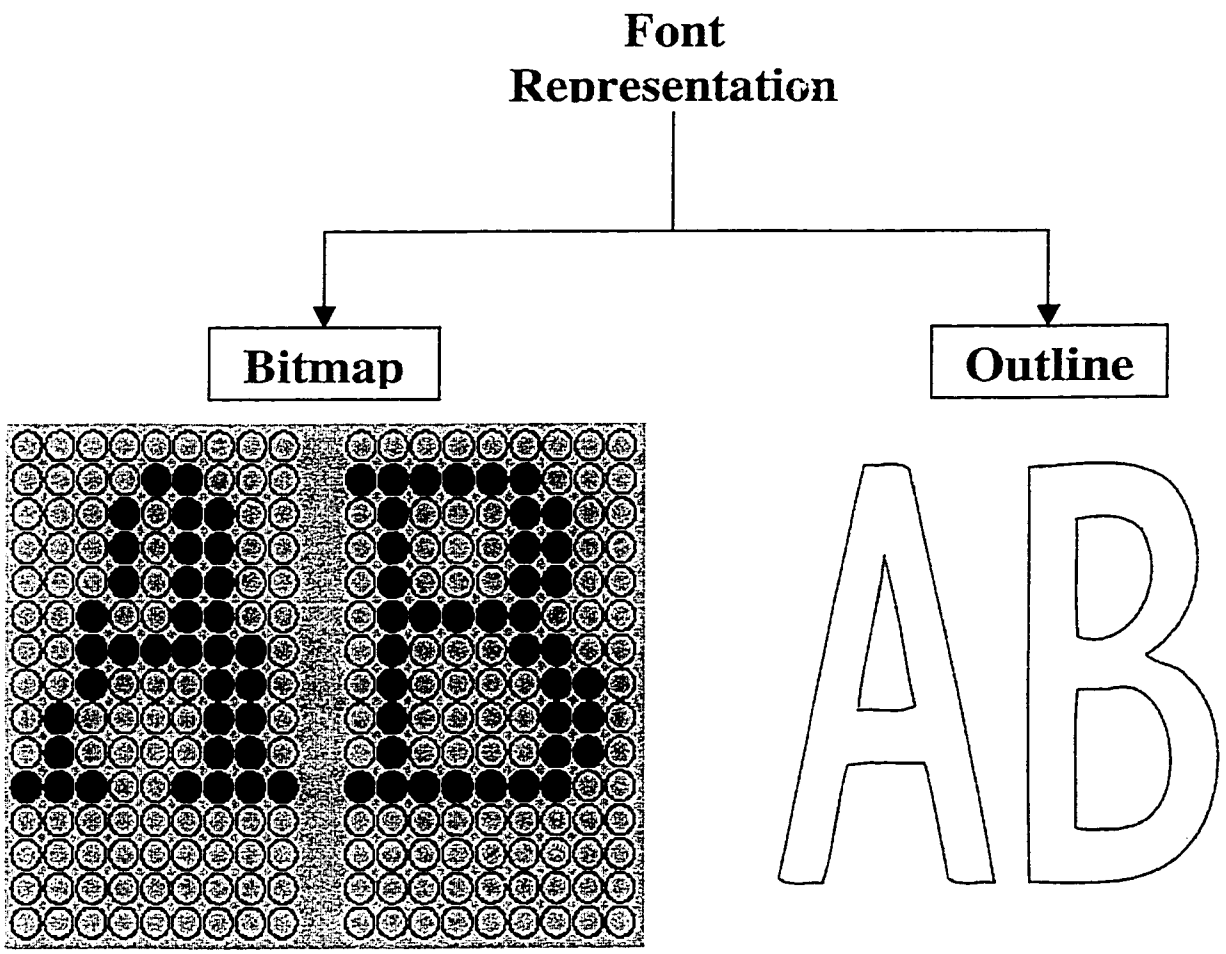


Figure 1.1: Font Representation Methods

For example one pixel per bit image has only two possible values of bit i.e. bit is either on (1) or off (0). Thus it has only two possible colors (black or white). Bitmap graphics are referred as raster graphics.

In this method each character in a given font is specified as a small rectangle bitmap. Information about height of the characters and the amount of space to be placed between adjacent characters is also stored. Generating a character requires to copy those pixels values from font cache into the frame buffer and then on to the screen at the desired position.

The bitmaps for the font cache are usually created by scanning in enlarged pictures of characters from typesetting fonts in various sizes. A paint program is used to specify the individual pixels in each character's bit map. Alternatively, the type designer can use a paint program to create fonts from scratch. Since small bitmaps do not scale well, it is necessary to define more than one bit map for each character to provide various standard sizes.

1.4.2 Outline

Fonts can be represented by outline using spline. Splines are piecewise polynomial parametric curves, generated by varying a parameter over a specified range. If $x(t)$ and $y(t)$ are two functions that supply points $(x(t), y(t))$ along a curve as t is varied then mathematically we can write:

$$x(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n$$
$$y(t) = b_0 + b_1 t + b_2 t^2 + \ldots + b_n t^n \tag{1.1}$$

where a_i and b_i are coefficients and n is the order of the polynomial

In equation (1.1) when n is 2, the polynomials are called *conic*; when n is 3, they are *cubic*. To approximate a particular curve, it is break into segments that meet at their end points. The meeting points are called joint points or significant points. A pair of formulas defines each segment in the spline. The coefficients are not chosen arbitrarily, but rather so as to achieve smoothness at the joints. In general splines of order n have continuity in the $(n-1)$ derivatives at each join point. Therefore conic spline has continuous first derivative (slope), and cubic has continuous first and second derivatives at the joints.

In the outline representation characters are represented in an abstract device independent way. Piecewise polynomial functions i.e. splines provide smooth curves. These splines are used to store the outline of characters. Although each character definition may take more space than its representation in a font cache but multiple sizes may be derived from a single stored representation by suitable scaling. Different type faces can also be derived e.g. italics may be derived by shearing the original outline. Another advantage of storing characters in a device independent form is that the outlines may be arbitrarily translated, rotated, scaled, and clipped.

1.4.3 Drawbacks of Bitmap Representation of Fonts

- It requires a distinct font cache for each combination of font, size, and face for each different resolution of display or output device. For example a single font in six different point sizes and four faces (normal, bold, italic, bold italic) requires 24 font caches.
- Bitmap representation of font behaves unpredictably when subject to Affine Transformations, thereby giving an unacceptable shape and resolution.
- Bitmap fonts always appear the same even if the resolution of device is enhanced

1.5 Forms of Outline Capture

Depending on the mathematical model of parametric function used to represent a curve, outline of fonts can be captured in two different ways:

1. Capture by Interpolation.
2. Capture by Approximation.

1.5.1 Capture by Interpolation

In this form of capture, parametric curve is constrained to pass through all the given set of data points. Lets for example t is our parameter, ranging from 0 to 1, and

for some values of t we have corresponding (x, y) values. These $(x,\ y)$ values are our data points. But for those values of t we do not have corresponding (x, y) values, we devise an interpolating function to find them. The technique of capturing outline of font parametrically by interpolation is suitable when the data points describing the digitized contour are sufficiently accurate and smooth.

1.5.2 Capture by Approximation

In this form of capture, parametric curve passes close to the given set of data points, but not necessarily through them. By some distance criteria, it is ensured that the overall shape of the font, described by given set of data points, is preserved. The distance (between captured curve and given data point) is commonly measured along a coordinate or along a normal to the captured curve. The distance value can be used in a variety of ways to achieve best-fitting approximation to given set of data points. It may be employed in a way that requires all the data points to be within a predefined tolerance limit. Alternatively, an acceptable approximation may be where the average of all the distance values is below some predefined tolerance level.

The technique of outline capture of font parametrically by approximation is suitable when the data points describing the digitized contour contain some erroneous values (e.g. noise). The function/curve that approximates the known values, on the average, minimize the total error. This idea is based on the statistical notion that errors in data are randomly distributed.

In our method of font design we have chosen capture by approximation technique. There are several reasons for this decision.

1. When the original image on paper is scanned to get its bitmap representation, there is always a possibility that all the features of original image are not preserved. Because limitation of scanner scanning capability and inter pixel gap problem. To understand the inter pixel gap problem lets consider a situation that a delicate curve passes through gap between pixels. Then its bitmap representation will select a pixel and that selection may not truly reflect the font designer choice. Therefore to design fonts of smooth outline, capture by approximation is better choice.

2. Extracted contour or boundary has noise (jagged edges). In this situation capture by interpolation will require a large number of data points to be saved and outline will not be smooth. Therefore, capture by approximation not only require very few data points but it gives smooth outline fonts.

3. Another advantage of capture by approximation technique is that we approximate the original contour up to certain tolerance level and if this tolerance level is zero then it is interpolation. So we have the maximum flexibility in our method.

In our method parametric curve always passes through significant points, regardless of tolerance limit between digitized curve and parametric curve. So for

significant points it is always interpolation.

1.6 Organization of Thiesis

chapter 1 is introductory chapter. Chapter 2 is about Parametric Curves. It covers various kinds of curves and some historical details how they are used in curve fitting. Chapter 3 is about Corner Detection. Corner Detection is a very important step in our method so details about our algorithm of corner detection are given in this chapter. Chapter 4 is most important chapter of our work. It describes our Font Design System in detail. Chapter 5 evaluates the system. Chapter 6 is related to Transformations and Mapping. Chapter 7 concludes our work and gives suggestion for future work.

Chapter 2

PARAMETRIC CURVES

Representing fonts in parametric form essentially requires to understand the underlying theory of parametric curves. In this chapter we will describe various kinds of curves as it will give reader a comparative view and help him to justify our choice of using cubic Bezier for capturing outline of fonts.

2.1 Parametric Representation of Curves

In this section we will discuss need and significance of parametric representation of curves. Our main references of this section are [10] and [11]. Polylines are first-degree, piecewise approximation to curves. But if the curves being approximated are not piecewise linear then a large number of endpoint coordinates must be created and stored to achieve reasonable accuracy. This is expensive in terms of both com-

putation and space. Higher degree functions can be use to approximate the desired curves. The higher-degree approximations can be based on one of three methods. (We restrict our discussion to 2-D curves.)

- Explicit Functions
- Implicit Equations
- Parametric Representation

2.1.1 Explicit Functions

We can express y as explicit function of (e.g. $y = f(x)$). The difficulties with this approach are (1) it is not possible to get multiple values of y for a single value of x, so curves such as circle and ellipse must be represented by multiple curve segments; (2) such curves are not rotationally invariant and may require breaking a curve segment into many segments; (3) describing curves with vertical tangents is difficult, because a slope of infinity is difficult to represent. Figure(2.1) shows the graph of explicit function $y = sin(x)$.

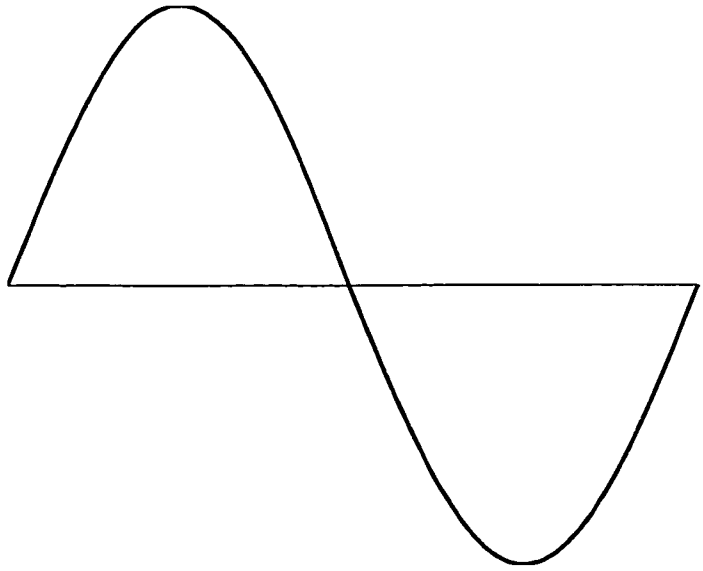


Figure 2.1: Sine Function

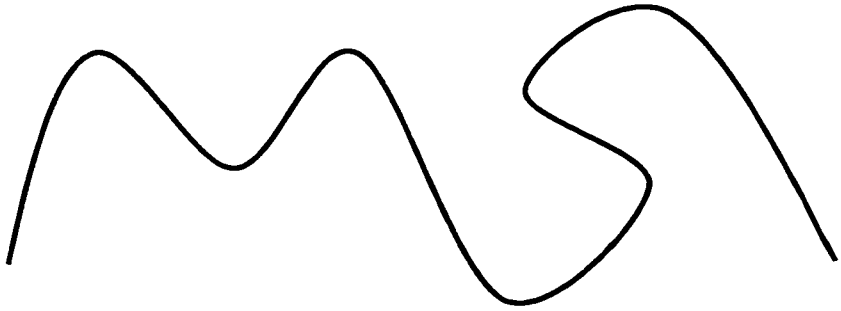


Figure 2.2: An Arbitrary Curve

2.1.2 Implicit Equations

We can choose to model curves as solutions to implicit equations of the form $f(x, y) = 0$; The difficulties with this approach are: (1) the given equation may have more solutions than required, for example, in modeling a circle, we might want to use $x^2 + y^2 = 1$, which is fine. But how do we model one-half of a circle? We must add constraints such as $x \geq 0$ which cannot be contained within the implicit equation; (2) if two implicit defined curve segments are joined together, it may be difficult to determine whether their tangent directions agree at join point.

2.1.3 Parametric Representation

The parametric form overcomes the problems caused by functional and implicit forms. The points on a curve are represented as ordered set of values: $p_i = [x_i, y_i]$. There are corresponding parametric functions that may be used to represent arbitrary curves; these are of the form $Q(t) = [x(t), y(t)]$. The parameter t takes the values from a specified range; conventionally from 0 to 1. A curve represented in this way can be thought of as the projection of the curve in 3-D, t as the third dimension, perpendicular to x and y plane. A circle can be represented in parametric form $Q(t) = [cos(t), sin(t)]$. Parametric form of curve allows multiple values of y for single or more values of x. Parametric curves replace the use of geometric slopes (which may be infinite) with parametric tangent vectors (which are never infinite).

A parametric curve is approximated by *piecewise polynomial curves.* Cubic polynomials are most often used because lower-degree polynomials give little flexibility in controlling the shape of the curve, and higher degree polynomials require more computation and can introduce unwanted wiggles. No lower-degree polynomials allow a curve segment to interpolate (pass through) two specified end points with specified derivates at each end point.

2.2 Overview of Curves and their Applications in Curve Fitting

This section describes various kinds of parametric curves and for each curve we will discuss past approaches of curve fitting [10],[11] and [29]. Bezier curves are covered in details as we used cubic Bezier curves for our font design method.

2.2.1 Spline

The term spline goes back to the long flexible strips of metal used by draftsmen to lay out the surfaces of ships, cars, aircrafts etc. Weight attached to splines called "ducks", were used to pull the spline in various directions. These metal splines had second-order continuity. Figure (2.3) shows draftmen's spline. The mathematical equivalent of these strips, the natural cubic spline, is a C^0, C^1, and C^2 continuous cubic polynomial that interpolates the control points.

A huge amount of work has been done towards the construction of splines in the last two decades. Various families of splines for different objectives have been discovered.

Figure 2.3: Draftmen's Spline

Curve fitting by Spline

One early reference to use of splines for approximation of functions appears in [26]. This method applied to single-valued functions.

Given a set of points (x_i, y_i) , $i = 0, \ldots, n$ construct a function $f(x)$ to minimize

$$\int_{x_0}^{x_n} g''(x^2)dx \tag{2.1}$$

among functions $g(x)$ such that

$$\sum_{i=0}^{n} \left[\frac{g(x_i) - y_i}{\delta y_i} \right]^2 \leq S \quad g \in C^2[x_0, x_n] \tag{2.2}$$

(Note that $S = 0$ yields interpolation)

Because of the use of second derivative in equation (2.2), Reinsch used cubic splines.

Grossman [14] described a parametric curve fitting technique. He noted that ordinary least-square fitting method does not take into account the ordering of points, and can not handle the multi-valued functions. His approach is to treat the data points parametrically:

$$\begin{aligned} x' &= f_x(u) \\ y' &= f_y(u) \end{aligned} \tag{2.3}$$

Assigning parametric values u by relative chord length

$$L_n = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{2.4}$$

If L_n is the chord-length of the digitized curve we are attempting to fit by parametric curve. Then, each point p_k is assigned a u-value (its parametric value) of L_k/L_n. Grossman's algorithm can be described as follows:

1. Parameterize the digitized curve using chord-length method.

2. Use least-square method independently in x and y, generating a polynomial curve.

3. Improve the parameterization of each (x_i, y_i).

4. Repeat step 2 and step 3 until a satisfactory fit is obtained.

Grossman's method for reparameterization is as follows: the contribution of the ith point to the total squared error is:

$$s(u_i) = w_{x_i}\left[x_i - f_x(u_i\right]^2 + w_{y_i}\left[y_i - f_y(u_i\right]^2 \tag{2.5}$$

where w_{x_i} and w_{y_i} are the weights assigned the errors associated with point (x_i, y_i). Therefore a new parameter u_i' is determined for each point (x_i, y_i) such that $s(u_i') \leq s(u_i)$. The Grossman's approach uses constant step-size search. The position is refined by parabolic interpolation, and the process is then repeated with a smaller step-size.

This technique is also the basis of curve fitting systems in [24] and [29]. It is worthy to mention that Grossman's method [14] is not a piecewise representation. In order to fit complex curves, higher-degree polynomials are required.

2.2.2 B-Spline

B-splines consist of curve segments whose polynomial coefficients depend on just a few control points. This behavior is called *local control.* Thus, moving a control affects only a small part of curve. This local behavior is due to the fact that each vertex is associated with a unique basis function. The B-spline basis allows the order of basis function and hence the degree of the resulting curve to be changed without changing the number of defining polygon vertices.

Letting $P(t)$ be the position vectors along the curve as a function of the parameter t, a B-spline curve is given by

$$P(t) = \sum_{i=1}^{n+2} B_i N_{i,k}(t) \quad t_{min} \leq t < t_{max}; \;\; 2 \leq k \leq n+1 \tag{2.6}$$

where B_i are the control points(position vectors) of the $n+1$ defining vertices and $N_{i,k}$ are the normalized B-spline basis functions. For the ith normalized B-spline basis function or order k (degree $k-1$) is defined by the Cox-deBoor recursion formulas.

$$N_{i,1} = \begin{cases} 1 & \text{if} \;\; x_i \leq t < x_{i+1}; \\ 0 & \textit{otherwise.} \end{cases}$$

and

$$N_{i,k}(t) = \frac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \tag{2.7}$$

In above equations, x_i are *knots* , satisfying the relation $x_i \leq x_{i+1}$. The parameter t varies from t_{min} to t_{max} along the cure $P(t)$. Since the denominators in the recursive calculations can have a value 0, the convention 0/0=0 is assumed.

Formally a B-spline curve is defined as a polynomial spline function of order k (degree $k-1$). $P(t)$ and its derivatives of order $1, 2, \ldots, k-2$ are all continuous

over the entire curve. Thus, for example, a fourth-order B-spline curve is a piecewise cubic curve (C^2 continuity).

From equation (2.7) it is clear that the choice of knot vector (x_i) has a significance influence on the B-spline basis function $N_{i,k}(t)$ and hence on the resulting B-spline curve. Fundamentally three types of knot vectors are used: *uniform, open-uniform (or open) and non-uniform.*

Curve fitting by B-spline

If data points and knot values are known then control points can be computed and thereby B-spline curve can be fitted. This is know as inversion method.

Wu, Abel and Greenburg [34] used B-spline for surface representation. The user of the system has to choose knots from a digitized curve then by inversion method B-spline curve is approximated. Later user can modify control points to alter shape of parametric curve. This method involves intervention of user in curve fitting technique as opposed to our method.

Yamaguchi [35] developed a method of an interactive curve fitting using B-spline. In his method the designer sketches a curve (digitized curve) and then picks candidate knots from the sketched curve. The developed system selects a subset of approximately equidistant candidates as knots. Then by inversion process parametric B-spline curve is approximated. User may move knots interactively to improve fit. If the fit is unsatisfactory then knots are automatically increased and system

again tries to fit the curve. This system also requires intervention of user.

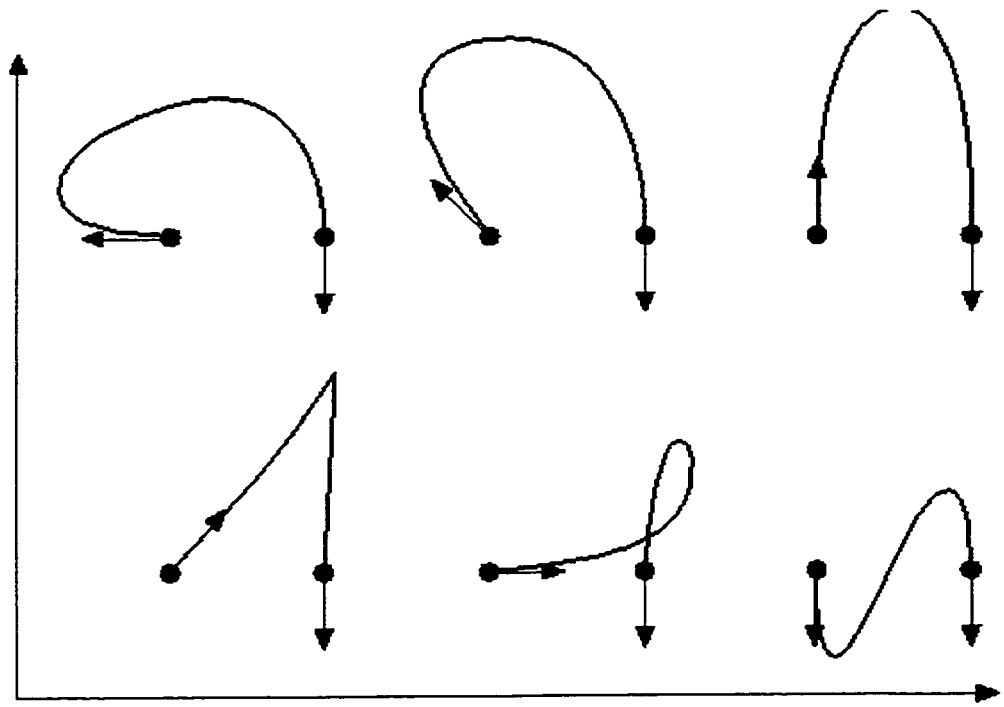
A least square cubic B-spline curve fitting technique is described by Dierchx [6]. A cubic polynomial is fitted to the curve with no interior knots. If the error exceeds tolerance level, then number of knots are increased. This process is repeated until a good fit is obtained.

A curve fitting technique based on quadratic B-splines is described by Yang [36]. In his method knots are chosen from digitized curve and parabola is fitted to sequence of points. If error exceeds specified threshold at some point then previous point is taken as knot vector, and the method is repeated to new set of knots. After having final set of knot vectors, quadratic B-spline is fitted to the piecewise parabola.

Vercken [33] described a system based on B-spline. In his system user draws a digitized curve by mouse. Points are parameterized using chord-length parameterization. Two least square polynomials are fitted to the curve. It the distance from digitized points to the parametric curve is greater than some tolerance, then the curve is subdivided into k pieces, where k is the order of spline. The division points chosen in this manner are knot vectors for the next stage. In next stage, B-spline is fitted to the curve, using knot vectors chosen in previous stage.

2.2.3 Hermite

Hermite curves are defined by two points P_1 and P_4 and two tangent vectors R_1 and R_4. Figure (2.4) shows a Hermite curve.



● End Points P_1 and P_2
→ Tangent Vectors T_1 and T_2

Figure 2.4: Hermite Curves

To find *Hermite basis matrix* M_H, which relates the Hermite geometry vector G_H to the polynomial coefficients, we write four equations, one for each of the constraints (P_1, P_4, R_1, R_4), in the four unknown polynomial coefficients, and then solve for the unknowns.

Defining G_{H_x} , the x component of the Hermite geometry matrix, as

$$G_{H_x} = \begin{bmatrix} P_{1_x} & P_{2_x} & P_{3_x} & P_{4_x} \end{bmatrix} \tag{2.8}$$

We express the curve as:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x && (2.9) \\ &= G_{H_x}.M_H.T \\ x(t) &= G_{H_x}.M_H \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} && (2.10) \end{aligned}$$

The constraints on $x(0)$ and $x(1)$ are found by substituting $t = 0$ and $t = 1$ in equation (2.10)

$$x(0) = P_{1_x} \;=\; G_{H_x}.M_H \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{2.11}$$

$$x(1) = P_{4_x} \quad = \quad G_{H_x}.M_H \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T \tag{2.12}$$

To get $x'(t)$ we have to differential equation (2.9)

$$x(t) \quad = \quad G_{H_x}.M_H \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \tag{2.13}$$

Now we can find $x'(0)$ and $x'(1)$

$$x'(0) = R_{1_x} \quad = \quad G_{H_x}.M_H \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T \tag{2.14}$$

$$x'(1) = R_{4_x} \quad = \quad G_{H_x}.M_H \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix}^T \tag{2.15}$$

Now equation(2.8) can be written as follows

$$G_{H_x} \quad = \quad G_{H_x}.M_H \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.16}$$

M_H is inverse of 4x4 matrix

$$M_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.17}$$

M_H can now be substituted in (2.10) to find $x(t)$. Similarly $y(t)$ and $z(t)$ can be found out. So we can write for Hermite polynomial $Q(t)$

$$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} = G_H.M_H.T \tag{2.18}$$

where

$$G_H = \begin{bmatrix} P_1 & P_4 & R_1 & R_4 \end{bmatrix}$$

Hermite blending function B_H can be written as

$$B_H = M_H.T \tag{2.19}$$

Now $Q(t)$ can be written as follows

$$Q(t) = G_H.B_H$$

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4 \quad (2.20)$$

These basis functions are shown in Figure(2.5)

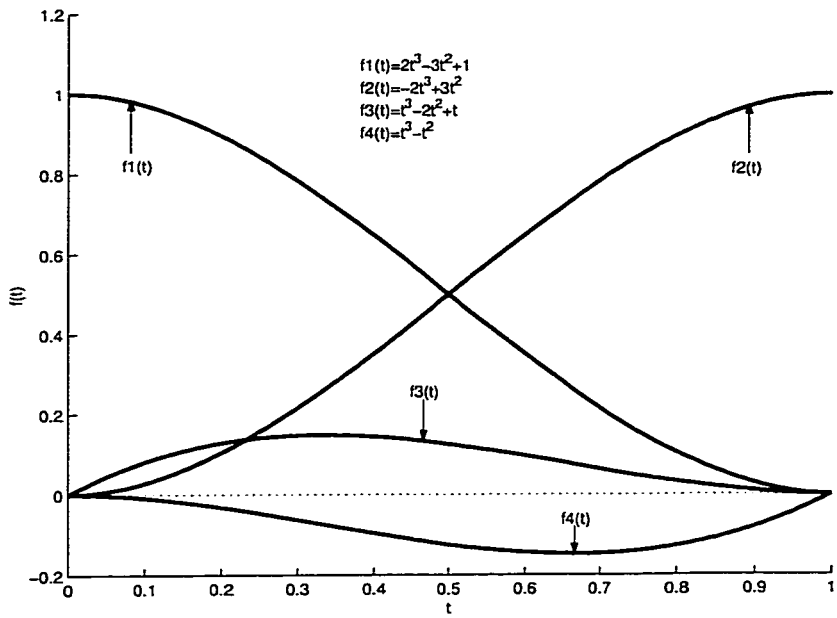


Figure 2.5: Hermite Basis Functions

Curve fitting by Hermite

Ichida [16] described curve fitting by one pass method. His method produces curve of C^1 continuity. Chong [3] described an automatic curve fitting algorithm, produces C^1 continuity. It is worth that his technique is linear.

2.2.4 Bezier

Bezier curves are developed by Pierre Bezier [22] and [23] for use in designing automobiles at Renault.

The Bezier form of cubic polynomial curve segment has four control points P_0, P_1, P_2 and P_3. Two intermediate points P_1 and P_2 are not on the curve. The Bezier curve interpolates the two end control points P_0 and P_3 and approximates the two intermediate points P_1 and P_2. Some typical Bezier curves and their control polygon are shown in Figure(2.6)

The starting and end tangent vectors R_0 and R_3 are determined by P_0P_1 and P_2P_3 as follows:

$$R_0 = Q'(0) = 3(P_1 - P_0) \tag{2.21}$$

$$R_3 = Q'(1) = 3(P_3 - P_2) \tag{2.22}$$

The *Bezier geometry matrix* G_B is defined as

$$G_B = \begin{bmatrix} P_0 & P_1 & P_2 & P_3 \end{bmatrix} \tag{2.23}$$

The matrix M_{HB} defines the relation between *Hermite geometry matrix* G_H and

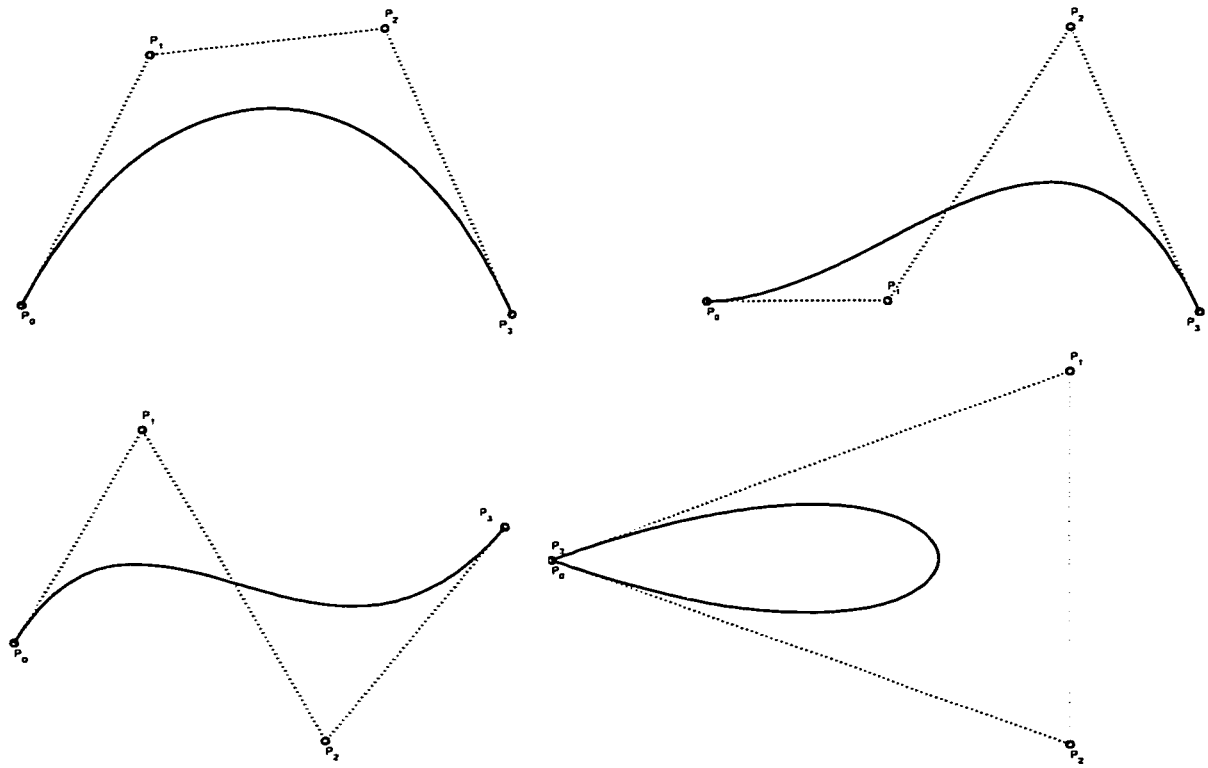


Figure 2.6: Bezier Curves and their Control Polygons

Bezier geometry matrix G_B:

$$
\begin{aligned}
G_H &= G_B.M_{HB} \\
&= \begin{bmatrix} P_0 & P_1 & P_2 & P_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix}
\end{aligned} \tag{2.24}
$$

To find *Bezier basis matrix* M_B, we use equation (2.18) for the Hermite form

$$
\begin{aligned}
Q(t) &= G_H.M_H.T \\
&= (G_B.M_{HB}).M_H.T \\
&= G_B.(M_{HB}.M_H).T
\end{aligned} \tag{2.25}
$$

we define

$$
M_B = M_{HB}.M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.26}
$$

Now we can write equation (2.25) as follows

$$Q(t) = G_B.M_B.T \tag{2.27}$$

Substituting the G_B, M_B and T yields

$$Q(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \tag{2.28}$$

Bernstein polynomials $B_0(t), B_1(t), B_2(t)$ and $B_3(t)$ can be defined as follows

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t(1-t)^2 \\ B_2(t) &= 3t^2(1-t) \\ B_3(t) &= 3t^3 \end{aligned} \tag{2.29}$$

Figure (2.7) shows Bezier basis functions.

General form of Bezier Bernstein polynomials

A Bezier curve of degree n is defined in terms of *Bernstein polynomials* as follows:

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{2.30}$$

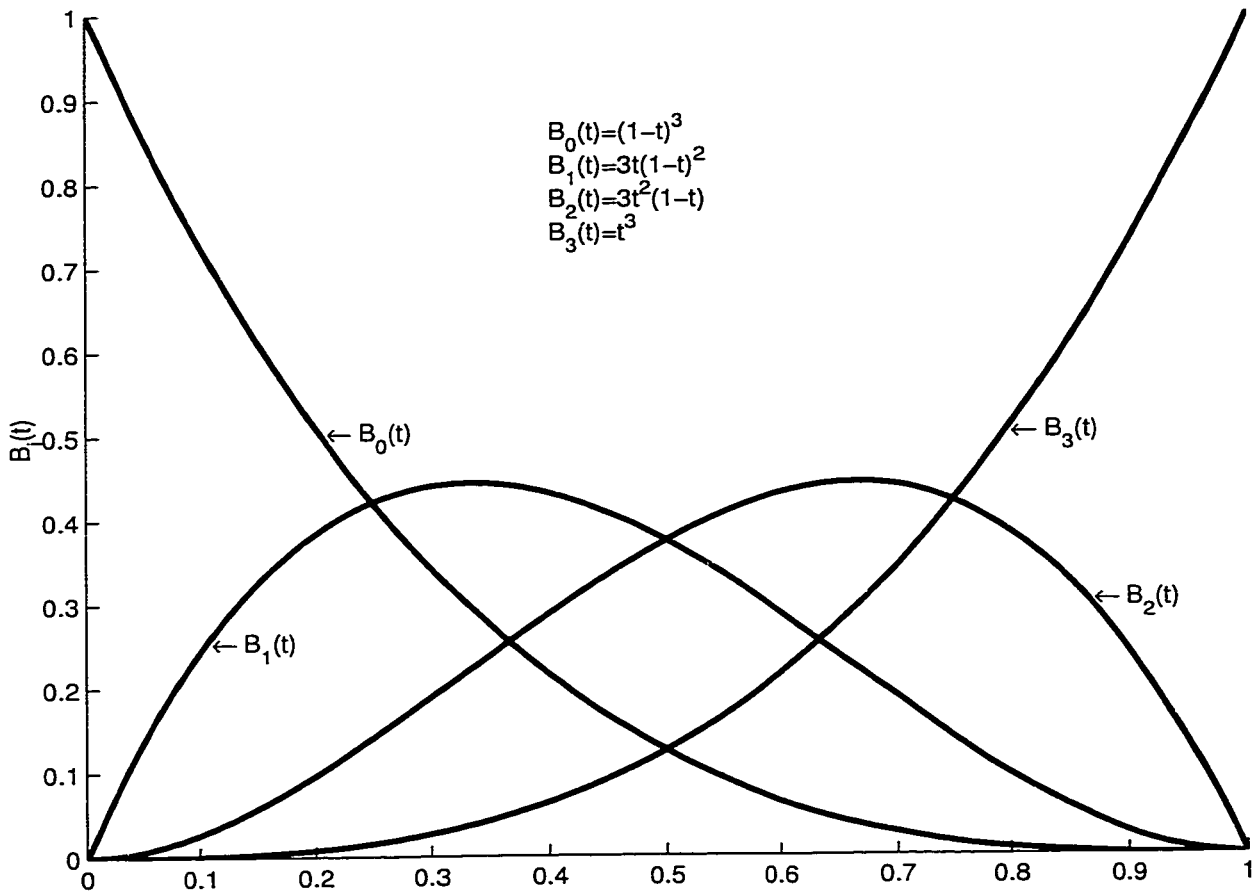


Figure 2.7: Bezier Basis Functions

$B_i^n(t)$ are *Bernstein polynomials*

$$B_i^n(t) = \left(\frac{n!}{i!(n-i)!}\right) t^i(1-t)^{n-i}, \quad i = 0, \ldots, n \tag{2.31}$$

P_i are the control points, and the sequence $P_0, P_2, \ldots, P_n$ form the control polygon of the curve segment.

De Casteljau Algorithm

To find the point $Q(t)$ on the curve given a particular t, a simple way is expanding the Bezier curve definition into a conventional form $f(t) = (x(t), y(t), z(t))$ and plugging a particular t into this equation to obtain $f(t)$. While it works, this method is not numerically stable (i.e., numerical errors will be introduced during the course of evaluating the polynomials). De Casteljau's elegant algorithm is designed for this purpose.

The fundamental concept of de Casteljau's algorithm is choosing a point **C** in line segment **AB** such that the distance between **A** and **C** and the distance between **A** and **B** has a given ratio, say t (i.e. $t = AC/AB$). Let us find a way to determine point **C**.

Figure 2.8: Choosing Point C in Line Segment AB

The vector from **A** to **B** is **B - A**. Since t is a ratio in the range of 0 (**C=A**) and 1

(**C**=**B**), point **C** is located at t(**B** - **A**). Taking the position of **A** into consideration, point **C** is **A** + t(**B** - **A**) = (1 - t)**A** + t**B**. Therefore, given a t, (1 - t)**A** + t**B** is the point between **A** and **B** such that the distance between it and **A** and the distance between **A** and **B** has a ratio t.

De Casteljau's construction is illustrated in Figure(2.9) for a Bezier curve with four control points.

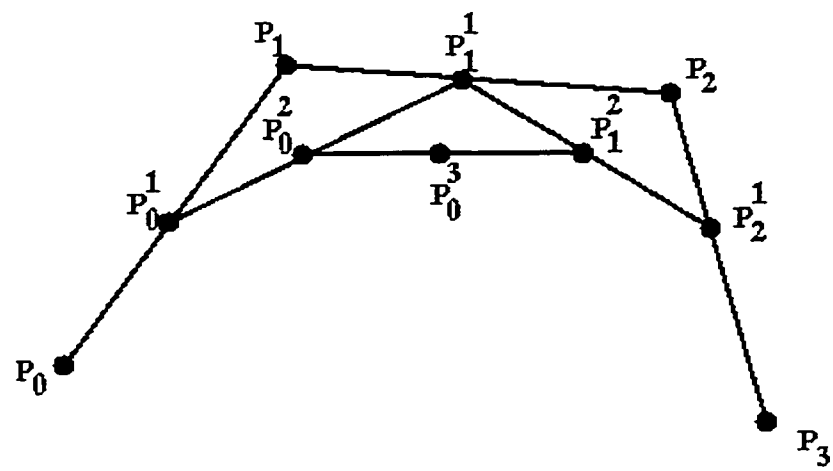


Figure 2.9: De Casteljau's consturction

The curve is defined by the control polygon with the four control points P_0, P_1, P_2 and P_3. Firstly for a given value of t the three points P_0^1, P_1^1 and P_2^1 are defined as the points with parameter t on each of the line segments P_0P_1, P_1P_2 and P_2P_3 using linear interpolation.

Now using linear interpolation on the two line segments $P_0^1P_1^1$ and $P_1^1P_2^1$ the two points P_0^2 and P_1^2 are produced.

Finally by linear interpolation on the line $P_0^2P_1^2$ the point P_0^3 is produced. This

point is on the cubic Bezier curve with the given control points. The process described here can be defined mathematically at the nth stage by:

$$P_i^0(t) = P_i \tag{2.32}$$

$$P_i^n(t) = (1-t)P_i^{n-1}(t) + tP_{i+1}^{n-1}(t) \tag{2.33}$$

Using the above recurrence with $n = 1$ gives:

$$\begin{aligned} P_0^1(t) &= (1-t)P_0 + tP_1 \\ P_1^1(t) &= (1-t)P_1 + tP_2 \\ P_2^1(t) &= (1-t)P_2 + tP_3 \end{aligned} \tag{2.34}$$

from which we calculate

$$\begin{aligned} P_0^2(t) &= (1-t)P_1^1 + tP_1^1 \\ P_1^2(t) &= (1-t)P_1^1 + tP_2^1 \end{aligned} \tag{2.35}$$

and finally

$$P_0^3(t) = (1-t)P_0^2 + tP_1^2 \tag{2.36}$$

If we expand the expression for P_0^3 using the previous recurrences until we have an expression only using the original control points we obtain the Bezier Cubic Polynomial:

$$Q(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

The algorithm, for computation, is designed as follows:

```
Input: array P of n+1 points and real number t
Output: point on curve, Q(t)
Working: point array q
for i := 0 to n do
    q[i] := P[i]; /*save input*/
endfor
for k := 1 to n do
    for i := 0 to n - k do
        q[i] := (1 -t)q[i] + tq[i + 1];
    endfor
endfor
return q[0];
```

A method for drawing closed rational curve specifed in terms of control points as two Bezeir segments is described by [13]. The method is based on de Casteljau algorithm.

Properties of Bezier Curve

Some of the important properties of Bezier curves are following:

Affine invariance: Bezier curves are invariant under affine maps. It means that we can rotate, translate, or scale the set of control points, and the curve associated with them will retain its relationship with the control points. Due to this property of Bezier curves we can rotate, translate, or scale generated fonts easily.

Convex hull: For$t \in [0, 1]$, the curve lies within the convex hull of control polygon. A simple consequence of the convex hull property is that a planar control polygon always generates a planar curve.

Endpoint interpolation: The Bezier curve passes through first and last control points. This can be easily verified by substituting $t = 0$ and $t = 1$ in Equation (2.28). We used this property in our font design system to ensure that curve passes through significant points of each segment.

Derivative of Bezier Curve: Derivatives of Bezier curve are used to enforce continuities. The kth derivative of Bezier curve is given by

$$\frac{d^kQ(t)}{dt^k} = \frac{n!}{(n-k)!}\sum_{i=0}^{n-k}\Delta^k P_i B_i^{n-k}(t) \tag{2.37}$$

where

$$\begin{aligned}\Delta^1 P_i &= \Delta P_i = P_{i+1} - P_i \\ \Delta^k P_i &= \Delta^k P_{i+1} - \Delta^{k-1} P_i\end{aligned}$$

In our font design method we determine first derivative of cubic Bezier for reparameterization. It means $n = 3$ and $k = 1$. We can write equation(2.37) as follows

$$\frac{dQ(t)}{dt} = Q'(t) = 3\sum_{i=0}^{2}(P_{i+1} - P_i)B_i^2(t) \tag{2.38}$$

$$Q'(t) = 3\left[(P_1 - P_0)B_0^2(t) + (P_2 - P_1)B_1^2(t) + (P_3 - P_2)B_2^2(t)\right] \tag{2.39}$$

From Equation (2.31) we can find the values of Bernstein polynomials

$$\begin{aligned}B_0^2(t) &= (1-t)^2 \\ B_1^2(t) &= 2t(1-t) \\ B_2^2(t) &= t^2\end{aligned} \tag{2.40}$$

Now we can write equation (2.39) as follows

$$Q'(t) = 3\left[(1-t)^2(P_1 - P_0) + 2t(1-t)(P_2 - P_1)) + t^2(P_3 - P_2)\right] \tag{2.41}$$

Chapter 3

CORNER DETECTION

Corners in digital images provide valuable information for shape representation and analysis. They provide important features for object recognition, shape representation, image interpretation and motion analysis [15], [4] and [7]. Therefore if these characteristic contour points are identified properly then a shape can be represented accurately and efficiently.

3.1 Corner Detection in Planar Curves

Corner detection is related to detection of high curvature points in planar curves. Various corner detection algorithms have been developed. Frequently cited approaches of corner detection are discussed and compared by [20] and [2]. For images a corner detection algorithm, based on the property of corners that the change of

image intensity should be high in all directions is described by [32].

What is corner? There is no generally accepted mathematical definition of corner. Rather it is intuitively understandable. Since we used curvature measure to detect corner, therefore in our case we set some threshold value of angle to declare a point as corner point.

In our font design method, corner points are the end control points (P_0 and P_3) of cubic Bezier curve. If corner points are detected accurately in early stage then computation will be minimize in the later stages.

A curve consists of sequence of points $p_i = (x_i, y_i), i = 1, 2, \ldots, n$. For each point its cornerness (measure of corner strength) is determined. Points whose cornerness is above predefined threshold are declared as corner points. When processing a point p_i, the algorithm considers a number of subsequent points (p_k^+) and previous points (p_k^-) in the sequence, as candidates for potential corner points.

3.2 Corner Detection Algorithm

The algorithm we used is a modified form of [2]. Corner point is defined as a point where triangle of specified angle can be inscribed within specified distance from its neighbor points. The number of neighbor points to be checked are also predefined.

First Pass: For each point p_i it is checked if triangle of specified size and angle

is inscribed or not. Following three conditions are used.

$$d_{min}^2 \leq |p - p_k^+|^2 \leq d_{max}^2 \tag{3.1}$$

$$d_{min}^2 \leq |p - p_k^-|^2 \leq d_{max}^2 \tag{3.2}$$

$$\alpha \leq \alpha_{max} \tag{3.3}$$

where

p is point under consideration for corner point.

p_k^+ is the kth clockwise neighbor of p.

p_k^- is the kth anti-clockwise neighbor of p.

Taking

$a = |p - p_k^+|$ $\quad$ $b = |p - p_k^-|$ $\quad$ $c = |p_k^+ - p_k^-|$

The angle α can be computed by using cosine law

$$a^2 + b^2 - c^2 - 2ab\,cos\alpha = 0 \tag{3.4}$$

$$\alpha = cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \tag{3.5}$$

All the three conditions described in equations (3.1), (3.2) and (3.3) are necessary for first pass. Now each point p may have zero, one or more than one alpha values. Among all alpha values, minimum value is taken as the alpha value of that point p.

Second Pass:The corner detection algorithm can detect adjacent points as corner. Because we specify threshold value of alpha and corner points as well as points neighbor to corners points can have value less than alpha. Second pass removes such points. If the difference of contour indices of corner points cp_i and cp_{i+1} is less than IndexLimit then the corner point whose alpha is high is removed from the candidate corner list. It means that if corner point cp_i has contour index i and corner point cp_{i+1} has contour index j then ($j - i \geq$IndexLimit).

The procedure of detecting corner points is given in the Flow chart of Figure (3.1), the pseudo codes are given in Figures (3.2) and (3.2).

Demonstration of the corner detection algorithm is made in Figures (3.4) to (3.13). We selected two figures to describe our experiments. One figure is of Arabic character "lillah" and the other figure is of English letter "S". Figure (3.4) shows the contour of "lillah" character and Figure (3.10) shows the contour of "S" character. Figure (3.5) to Figure (3.13) shows corner points obtained for various values of α. Table (3.1) summaries the results.

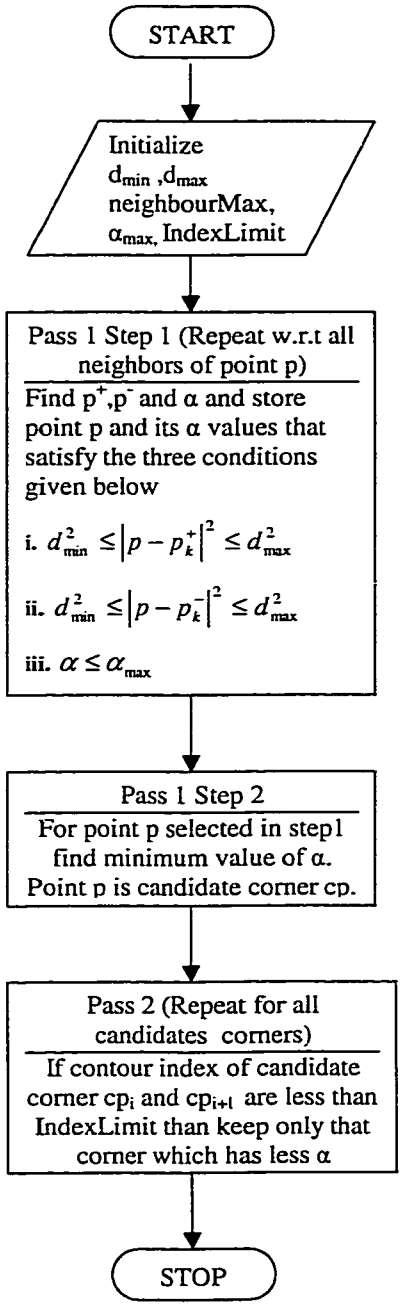


Figure 3.1: Flow Chart of Corner Detection Algorithm

```
Input: Points(p1, p2,...,pn) where pi=(px, py)
Output: Corner Points (cp1, cp2,...cpm) where cpi=(cpx, cpy)
alphaCount ← 0
dmin ←3
dmax ←dmin+3
neighbourMax← 8
αmax ← 150
IndexLimit ← 10

/* First Pass */
for p1Index ←1 to n
    for neighbourIndex←1 to neighbourMax
        if ((p1Index+ neighbourIndex)> n)
            p+Index← p1Index+ neighbourIndex-n
        else
            p+Index← p1Index + neighbourIndex
        end
        if ((p1Index- neighbourIndex)< =0)
            p-Index←n- neighbourIndex+ p1Index
        else
            p-Index← p1Index  - neighbourIndex
        end
        ds_pp+  ← ( px[p1Index]- px[p+Index] )^2  + ( py[p1Index]- py[p+Index] )^2
        ds_pp-  ← ( px[p1Index]- px[p-Index] )^2  + ( py[p1Index]- py[p-Index] )^2
        ds_p+p-  ← ( px[p+Index]- px[p-Index] )^2  + ( py[p+Index]- py[p-Index] )^2
        a ←sqrt(ds_pp+)
        b ←sqrt(ds_pp-)
        c ←sqrt(ds_p+p-)
        α ← cos^-1 [(a^2+b^2-c^2)/2ab]
        if ( ((ds_pp+ ≥ dmin) & (ds_pp+≤ dmax)) &
            ((ds_pp- ≥ dmin) & (ds_pp-≤ dmax))  &
            (α ≤ αmax ) )
            alphaCount← alphaCount+1
            α[p1Index, alphaCount]← α
        endif
    endfor  // End of neighbourIndex loop
        if (alphaCount > 0 )
            pass1Count← pass1Count+1 // count of first pass candidate corners
            // FindMinIndex function finds index of minimum α
            minIndex←FindMinIndex (α[p1Index, 1]... α[p1Index, alphaCount] )
            αmin [pass1Count]← α[p1Index, minIndex]
            cpIndex[pass1Count ]← p1Index // index of original contour points
            cpx[pass1Count]← px[p1Index]  // corner points after first pass
            cpy[pass1Count]← py[p1Index]  // corner points after first pass
        end
end  // End of p1Index loop
```

Figure 3.2: First Pass of Corner Detection Algorithm

```
/* Second Pass */

/* To remove very close corner points. Closeness is measured w.r.t contour index */
pass2Count← pass1Count
p2Index←2
while (p2Index<= pass2Count)
      if (cpIndex[p2Index] - cpIndex[p2Index-1] <IndexLimit)
        if (α_min [p2Index-1] > α_min [p2Index] )  // if true then replace
            α_min [p2Index-1]← α_min [p2Index]  // (p2Index-1)^th record
            cpIndex[p2Index-1]← cpIndex[p2Index] // with
            cp_x[p2Index-1]← cp_x[p2Index]  // p2Index^th record
            cp_y[p2Index-1]← cp_y[p2Index]
        end
        for shiftIndex←p2Index to pass2Count-1 // shift
            α_min [shiftIndex]← α_min [shiftIndex+1] // (p2Index+1) to pass2Count
            cpIndex[shiftIndex]← cpIndex[shiftIndex+1] //records
            cp_x[shiftIndex]← cp_x[shiftIndex+1] // one position up
            cp_y[shiftIndex]← cp_y[shiftIndex+1]
        end
        pass2Count←pass2Count-1  // last record in no longer corner record
        p2Index←p2Index-1
      end
      p2Index←p2Index+1 // proceed to next record
end
if (n- cpIndex[pass2Count]+ cpIndex[1] < IndexLimit) // check first and last records
     if  (α_min [1] > α_min [pass2Count]) // if true replace 1^st record with 2^nd record
        for shiftIndex←1 to pass2Count-1 // shift
        α_min [shiftIndex]← α_min [shiftIndex+1] // (1+1)^th to pass2Count
        cpIndex[shiftIndex]← cpIndex[shiftIndex+1] //records
        cp_x[shiftIndex]← cp_x[shiftIndex+1] // one position up
        cp_y[shiftIndex]← cp_y[shiftIndex+1]
    end
  pass2Count←pass2Count-1  // last record in no longer corner record
end

Corners( (cp_x[1], cp_y[1]), (cp_x[2], cp_y[2]),...,( cp_x[pass2Count], cp_y[pass2Count]))
```

Figure 3.3: Second Pass of Corner Detection Algorithm

Figure 3.4: Contour of Image

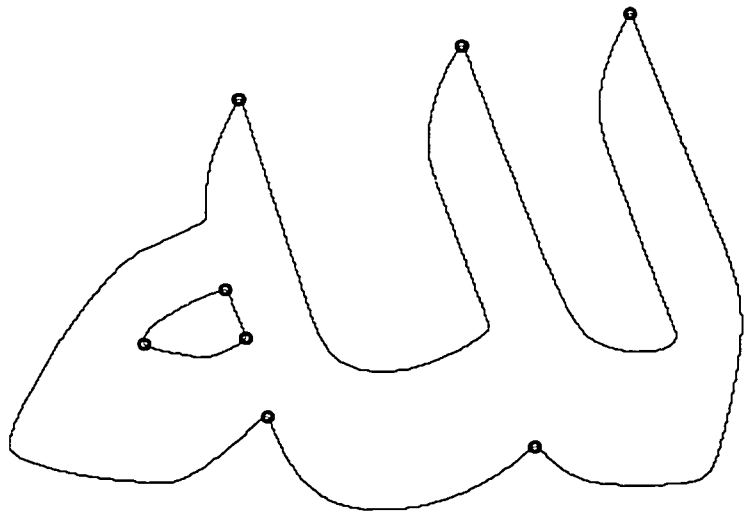


Figure 3.5: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 120$, $IndexLimit = 10$

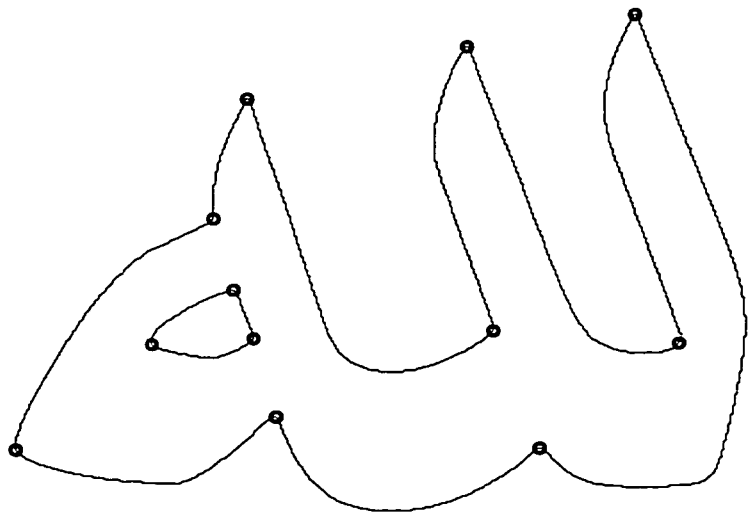


Figure 3.6: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 130 - 140$, $IndexLimit = 10$

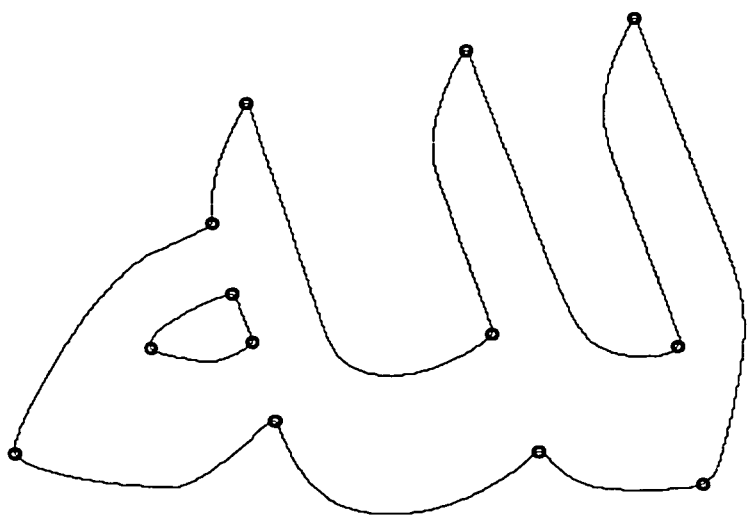


Figure 3.7: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 150$, $IndexLimit = 10$

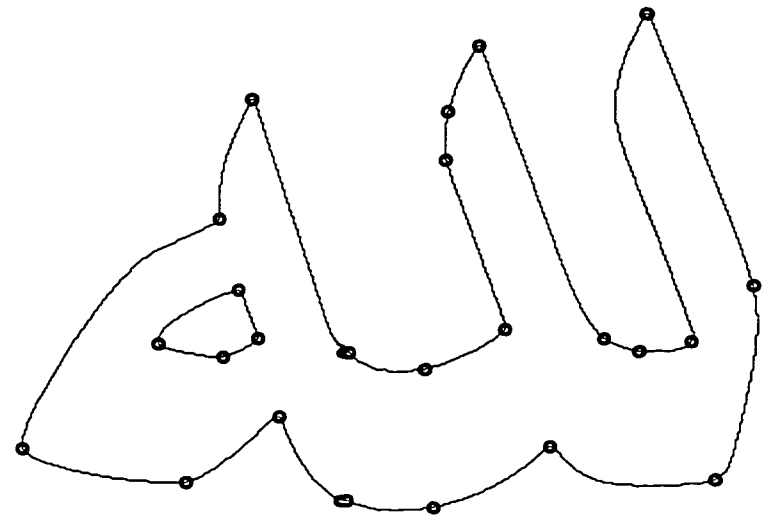


Figure 3.8: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 160$, $IndexLimit = 10$

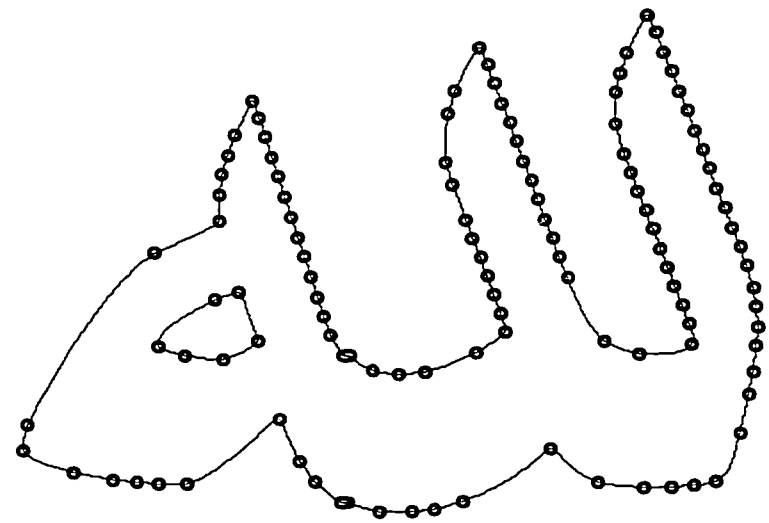


Figure 3.9: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 170$, $IndexLimit = 10$

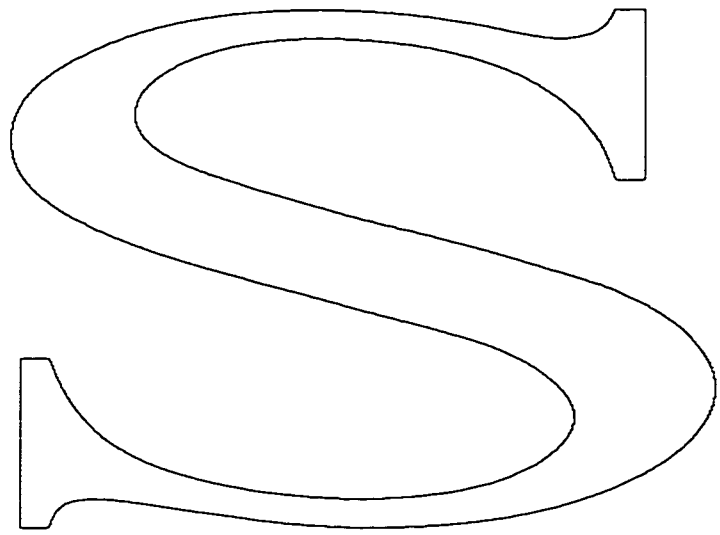


Figure 3.10: Contour of Image

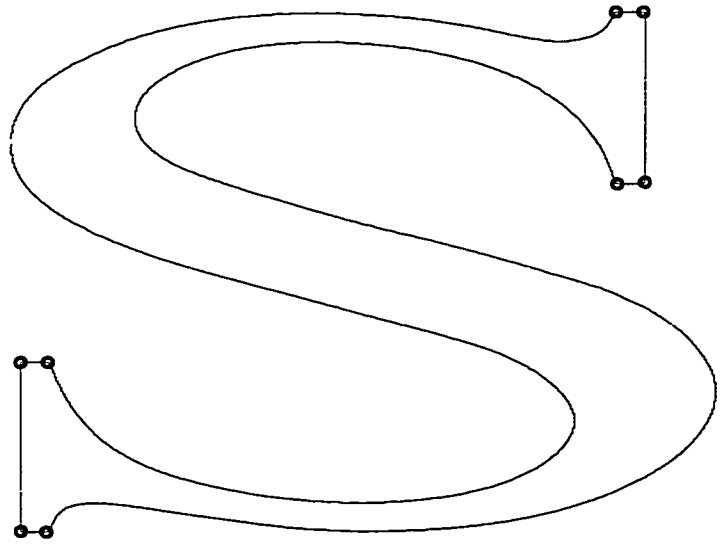


Figure 3.11: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 120 - 150$, $IndexLimit = 10$

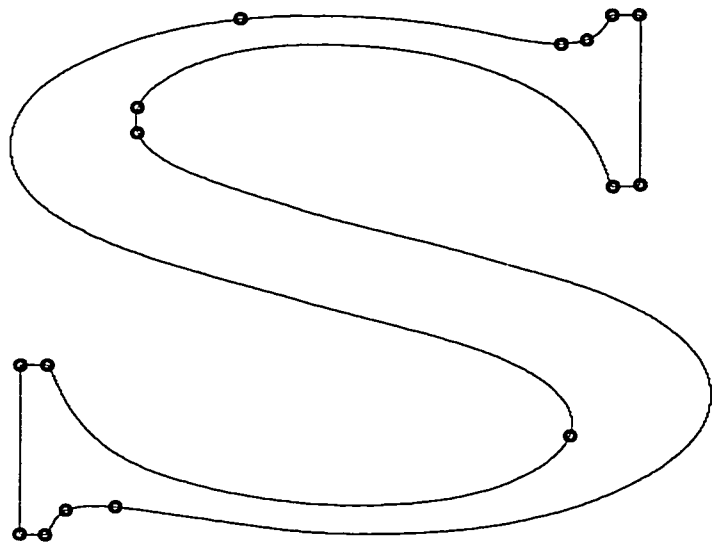


Figure 3.12: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 160$, $IndexLimit = 10$

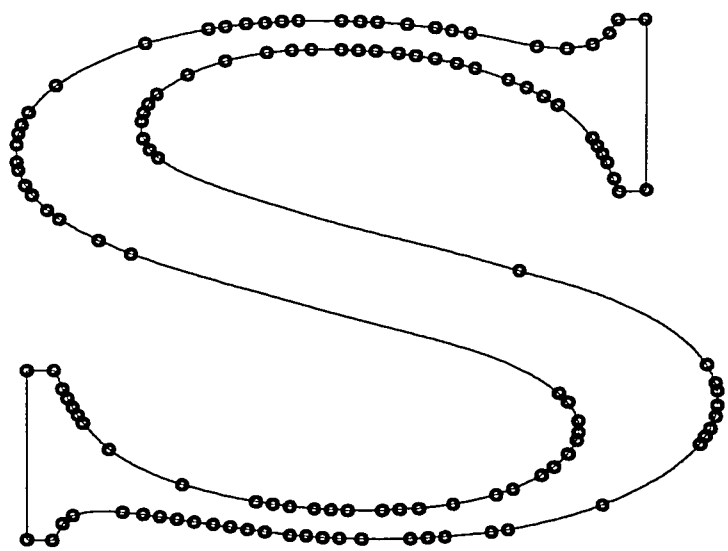


Figure 3.13: Corner candidates after Pass2. $d_{min} = 5$, $d_{max} = 8$, $\alpha_{max} = 170$, $IndexLimit = 10$

Table 3.1: Effect of α Value on Number of Corners. $d_{min} = 5$, $d_{max} = 8$, $NeighborMax = 8$, $IndexLimit = 10$

α_{max} (in degrees)	# of corners (Figure 3.4 image)	# of corners (Figure 3.10 image)
120	5+3	8
130	9+3	8
140	9+3	8
150	10+3	8
160	20+4	16
170	107+6	126

Chapter 4

FONT DESIGN SYSTEM

This chapter gives the details of our font design system. Our method automates the whole process of font design and consists of many steps. So it is necessary to understand each step in order to grasp the overall picture of the system.

The fundamental objective behind our system is to obtain the parametric representation of character that matches with its digitized image. Figure 4.1 shows the basic building blocks of our Font Design System. In this figure we did not show Noise Filtering and Reparmeterization steps. Later, we will describe how can we add these steps into our system to improve its performance. However the system can work without these two steps as well.

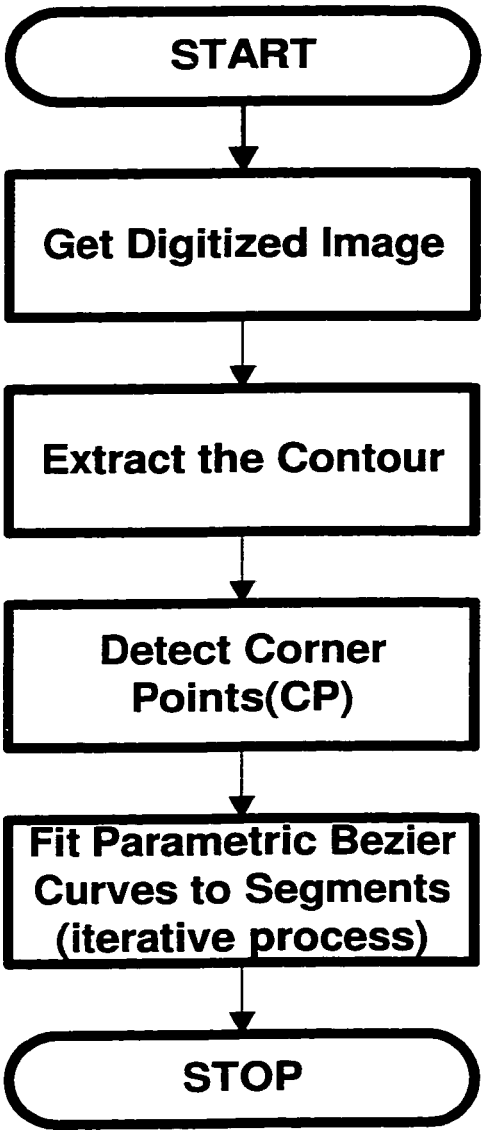


Figure 4.1: Building Blocks of Font Design System

4.1 Getting Digitized Image

Digitized image of a character can be obtained directly from some electronic device or by scanning an image. We used both methods. The quality of digitized scaned image depends on various factors such as image on paper, scanner and attributes set during scanning. The quality of digitized image obtained directly from electronic device depends on the resolution of device, source of image, type of image etc. Some of the digitized images are shown in Figures (4.2), (4.3) and (4.4).

4.2 Extraction of Contour

Contour of digitized image is extracted by using some boundary detection algorithm. There are numerous algorithms for detecting boundary. We used [25] algorithm. The input to this algorithm is a bitmap file. The algorithm returns number of pieces and for each piece number of boundary points and values of these boundary points $p_i = (x_i, y_i), i = 1, \ldots, N$. Since we are dealing with closed boundary curves therefore first and last boundary points of each piece have the same value. Figures (4.5), (4.6) and (4.7) show detected boundary of the images of Figures (4.2), (4.3) and (4.4) respectively. Table (4.1) summaries the results.

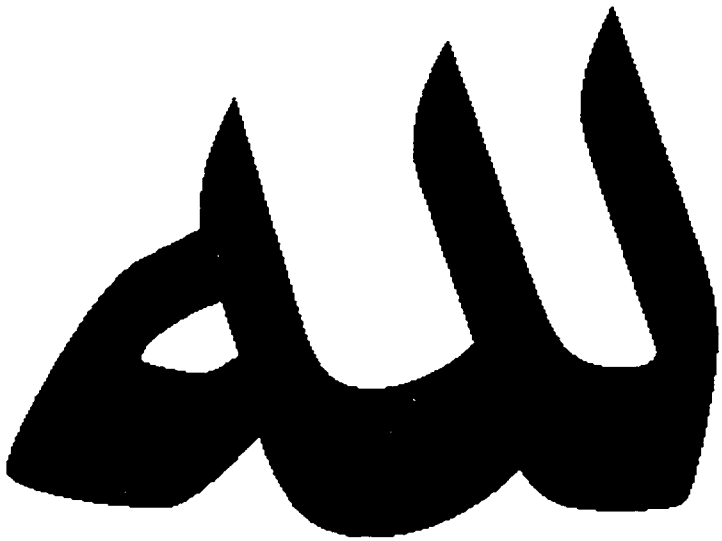


Figure 4.2: Digitized Image of 'Lillah' character

Figure 4.3: Digitized Image 'Ali' character

Figure 4.4: Digitized Image 'Samar' character

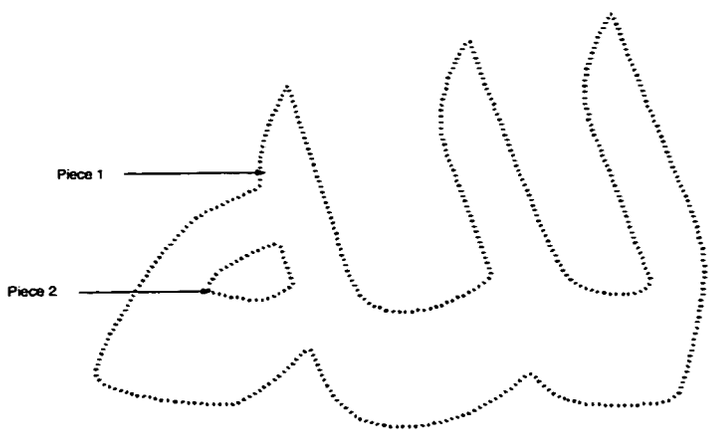


Figure 4.5: Detected Boundary consists of Two Pieces

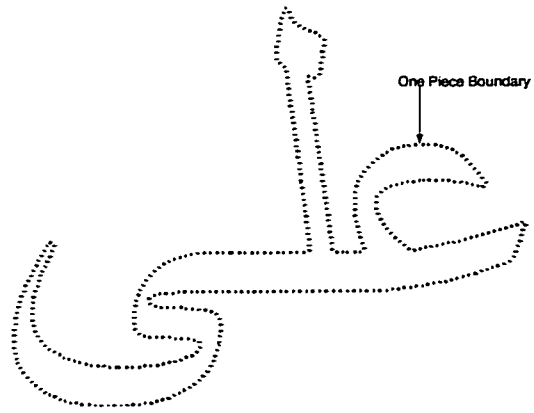


Figure 4.6: Detected Boundary consists of One Pieces

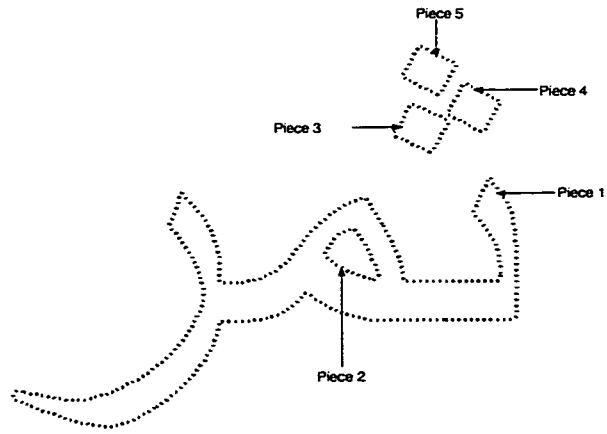


Figure 4.7: Detected Boundary consists of Five Pieces

Table 4.1: Contour Data

Figure #	# of Pieces	# of Boundary Points
4.5	2	1522+116=1638
4.6	1	1642
4.7	5	963+87+87+87+87=1311

4.3 Detecting Corner Points

Corner points are detected from the contour points,(see section §4.2) using curvature analysis technique. We already described the details of this process in chapter 3. We used 'o' to denote corner points. At this stage, significant points consist of only corner points but later on break points '△' will also be added in the set of significant points.

4.4 Fitting Parametric Cubic Bezier

We divide the contour points of each piece into groups called segments and fit cubic Bezier curve to each segment. The division is based on corner points. It means that if there are m corner points $cp_1, \ldots, cp_m$ then there will be m segments. For example first segment has all the contour points between corner point cp_1 and corner point cp_2 inclusive. Similarly second segment has all the contour points between corner point cp_2 and corner point cp_3 inclusive. Last segment has all the contour points between corner point cp_m and corner point cp_1 inclusive. Of course corner points obey the order of contour points. The situation is illustrated in Figure (4.8) for first piece of "Lillah" character.

If Contour Points of kth Segment are $p_u, \ldots, p_w$ then, we can opt the following notation.

$$p_u = cp_k \quad \text{where} \quad 1 \leq k \leq m \cdot$$

$$p_w = \begin{cases} cp_{k+1} & \text{if} \quad 1 \leq k < m; \\ cp_1 & \text{if} \quad k = m. \end{cases}$$

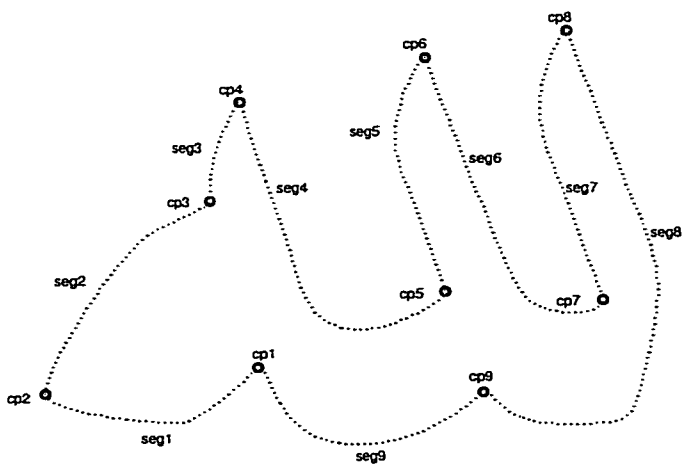


Figure 4.8: Division of Contour into Segments

For ease of manipulation we will assume that $p_u, \ldots, p_w$ there are n contour points in each segment such that $p_1 = p_u, p_2 = p_{u+1}, \ldots$ and $p_n = p_w$. The value of n of course may vary from segment to segment and can be computed as follows.

$$n = \begin{cases} w - u + 1 & \text{if} \quad u \leq w; \\ N - u + 1 + w & \text{if} \quad u > w. \end{cases}$$

where

N is total number of Contour points in a piece.

n is number of contour points in a segment.

Remember that we are dealing with closed boundary outline of characters. Each character has one or more pieces. Each piece have one or more segments.

Before going into details of fitting Cubic Bezier to segments, let us try to understand the problem. A parametric curve $Q(t)$ can be thought of as the projection of a curve in X, Y, t space onto $X - Y$ plane and our problem is to find the curve X, Y, t space whose projection approximates the digitized curve in *best* way. Infact curves defined by projections of 3-space curve on the $X - t$ and $Y - t$ planes are single value scalar curves. Therefore we have to devise a scheme that relates the X and Y coordinates of each point using parametric value t.

4.4.1 chord-length parameterization

We used *chord-length parameterization* to estimate the parametric value t associated with each point p_i. There are other possible ways to estimate the parametric value t as well. For example *unit parameterization* could be used. But we found *chord-length parameterization* more suitable and accurate. After having the t value associated with each point, we fit parametric cubic Bezier curve to set of data points of each segment.

$$t_i = \begin{cases} 0 & \text{if } \; i = 1; \\ \frac{|p_1p_2|+|p_2p_3|+,\ldots,+|p_{i-1}p_i|}{|p_1p_2|+|p_2p_3|+,\ldots,+|p_{n-1}p_n|} & \text{if } \; 2 \leq i \leq n-1; \\ 1 & \text{if } \; i = n \end{cases}$$

4.4.2 Finding Intermediate Control Points

As we have discussed in chapter 2, the Bezier form of cubic polynomial curve segment has four control points P_0, P_1, P_2 and P_3. Two intermediate points P_1 and P_2 are not on the curve. The Bezier curve interpolates the two end control points P_0 and P_3 and approximates the two intermediate points P_1 and P_2. The two end control points are the two corner points of the curve segment. But we have to find the two

intermediate control points of cubic Bezier to fit the curve.

$$\begin{aligned} Q(t) &= \sum_{k=0}^{3} P_k B_k(t) \\ &= (1-t^3)P_0 + 3t(1-t^2)P_1 + 3t^2(1-t)P_2 + t^3 P_3 \end{aligned} \tag{4.1}$$

Separating x and y terms we can write (4.1) as follows

$$Q_x(t) = (1-t^3)P_{x_0} + 3t(1-t^2)P_{x_1} + 3t^2(1-t)P_{x_2} + t^3 P_{x_3} \tag{4.2}$$

$$Q_y(t) = (1-t^3)P_{y_0} + 3t(1-t^2)P_{y_1} + 3t^2(1-t)P_{y_2} + t^3 P_{y_3} \tag{4.3}$$

where $0 \leq t \leq 1$, $P_0 = p_1$, $P_3 = p_n$

Recall that our goal is to approximate the digitized curve by a parametric curve in *best way*. For this purpose, we use the least square method. That is we define the sum of squared distances from the digitized curve to the parametric curve. Mathematically we can write:

$$
\begin{aligned}
S &= \sum_{i=1}^{n}[Q_i(t) - p_i]^2 \\
&= \sum_{i=1}^{n}[(Q_{x_i}(t) - p_{x_i})^2 + (Q_{y_i}(t) - p_{y_i})^2] \\
&= \sum_{i=1}^{n}[Q_{x_i}(t) - p_{x_i}]^2 + \sum_{i=1}^{n}[Q_{y_i}(t) - p_{y_i}]^2 \qquad (4.4)
\end{aligned}
$$

Our goal is to minimize S. We find partial derivatives of (4.4) with respect to P_1 and P_2 and equate them to zero. The solution will give us values of P_1 and P_2 that approximate the digitized curve by a parametric curve in *best way* for given values of t.

$$
\frac{\partial S}{\partial P_1} = 0 \qquad (4.5)
$$

$$
\frac{\partial S}{\partial P_2} = 0 \qquad (4.6)
$$

Expanding equation (4.5) yields

$$
\begin{aligned}
\frac{\partial S}{\partial P_1} &= 2\sum_{i=1}^{n}\frac{\partial Q(t_i)}{\partial P_1}[Q(t_i) - p_i] \\
&= 2\sum_{i=1}^{n}B_1(t_i)[Q(t_i) - p_i] \qquad (4.7)
\end{aligned}
$$

Expanding equation (4.6) yields

$$\begin{aligned}\frac{\partial S}{\partial P_2} &= 2\sum_{i=1}^{n}\frac{\partial Q(t_i)}{\partial P_2}[Q(t_i)-p_i] \\ &= 2\sum_{i=1}^{n}B_2(t_i)[Q(t_i)-p_i] \qquad (4.8)\end{aligned}$$

Let

$$\begin{aligned}A_k &= \sum_{i=1}^{n}[B_k(t_i)]^2 \\ A_{1,2} &= \sum_{i=1}^{n}B_1(t_i)B_2(t_i) \\ C_{x_k} &= \sum_{i=1}^{n}[B_k(t_i)[p_{x_i}-B_0(t_i)P_{x_0}-B_3(t_i)P_{x_3}] \\ C_{y_k} &= \sum_{i=1}^{n}[B_k(t_i)[p_{y_i}-B_0(t_i)P_{y_0}-B_3(t_i)P_{y_3}]\end{aligned}$$

Then solving (4.7) and (4.8) for P_1 and P_2 gives

$$\begin{bmatrix} A_1 & A_{1,2} \\ A_{1,2} & A_2 \end{bmatrix}\begin{bmatrix} P_{x_1} \\ P_{x_2} \end{bmatrix} = \begin{bmatrix} C_{x_1} \\ C_{x_2} \end{bmatrix} \qquad (4.9)$$

$$\begin{bmatrix} A_1 & A_{1,2} \\ A_{1,2} & A_2 \end{bmatrix} \begin{bmatrix} P_{y_1} \\ P_{y_2} \end{bmatrix} = \begin{bmatrix} C_{y_1} \\ C_{y_2} \end{bmatrix} \tag{4.10}$$

Solving (4.9) and (4.10) for intermediate control points $P_1 = (P_{x_1}, P_{y_1})$ and $P_2 = (P_{x_2}, P_{y_2})$

$$P_{x_1} = \frac{A_2 C_{x_1} - A_{1,2} C_{x_2}}{A_1 A_2 - A_{1,2}^2} \qquad P_{y_1} = \frac{A_2 C_{y_1} - A_{1,2} C_{y_2}}{A_1 A_2 - A_{1,2}^2} \tag{4.11}$$

$$P_{x_2} = \frac{A_1 C_{x_2} - A_{1,2} C_{x_1}}{A_1 A_2 - A_{1,2}^2} \qquad P_{y_2} = \frac{A_1 C_{y_2} - A_{1,2} C_{y_1}}{A_1 A_2 - A_{1,2}^2} \tag{4.12}$$

Now we have all the four control points $P_0, P_1, P_2,\ P_3$ and parametric value t. Using these control points and parameter value t we can fit the cubic Bezier to the segment. Fitted Cubic Bezier curves over digitized curves are shown in Figures (4.9), (4.10) and (4.11).

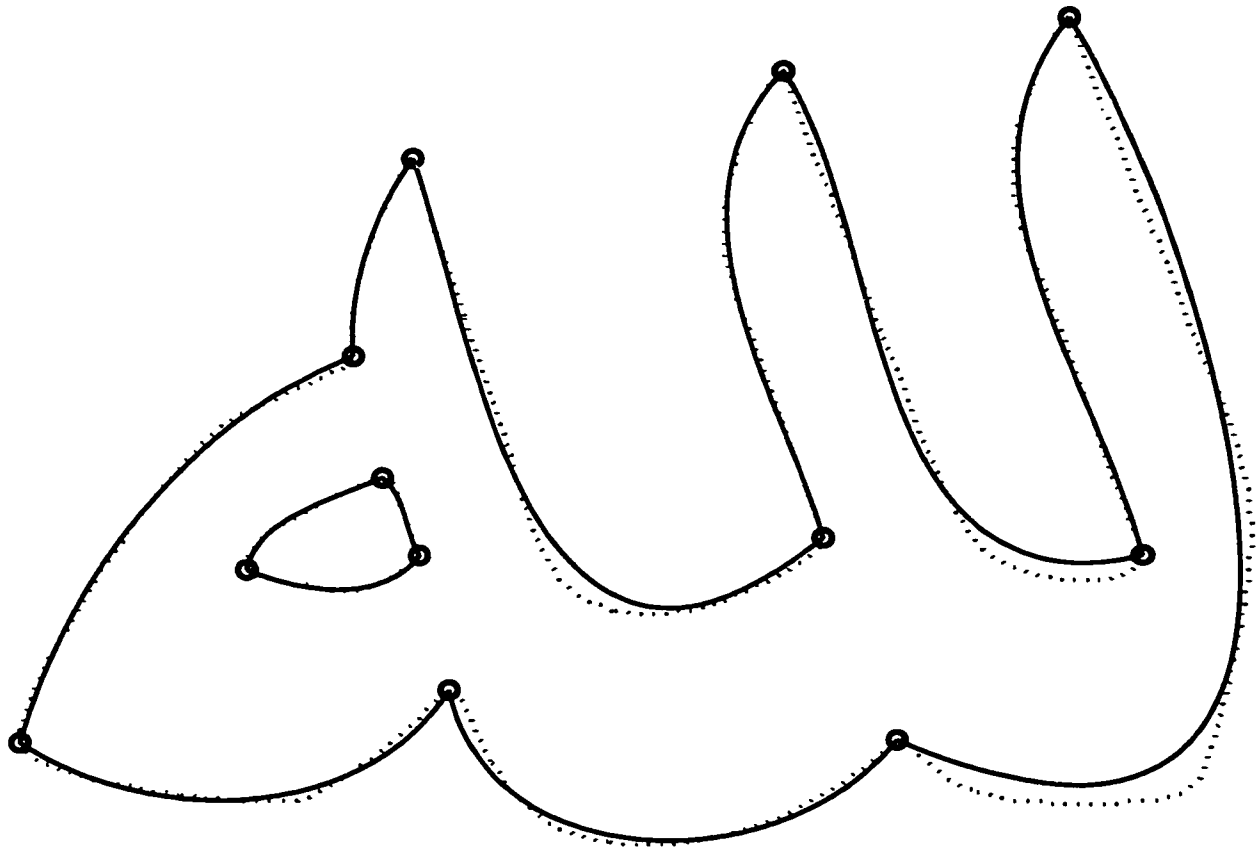


Figure 4.9: Fitted Cubic Bezier over Boundary

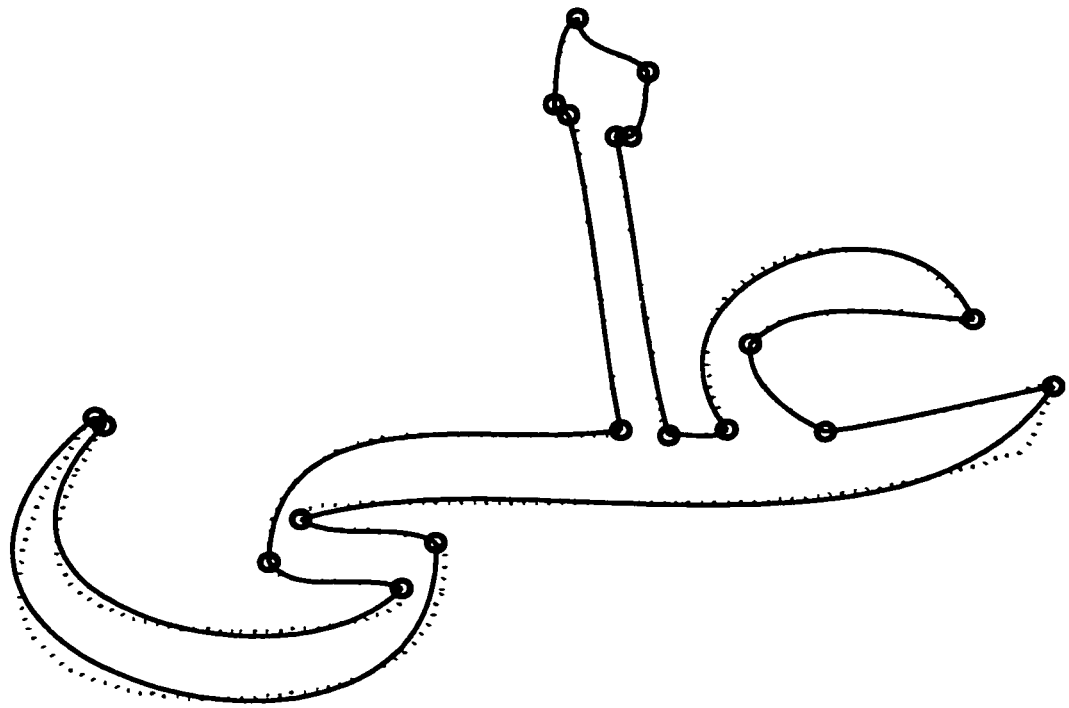


Figure 4.10: Fitted Cubic Bezier over Boundary

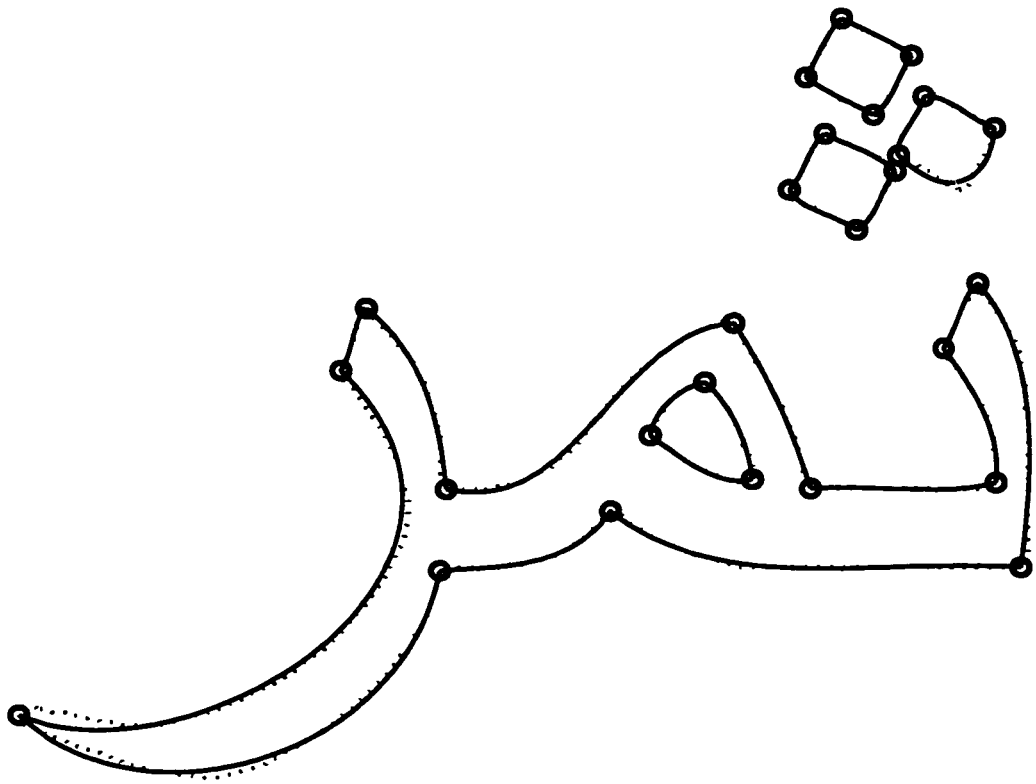


Figure 4.11: Fitted Cubic Bezier over Boundary

4.4.3 Breaking Segment

The distance between digitized curve and parametric curve should be within certain limit, we call this limit as threshold tolerance limit (d_{BP}).

The fitted Bezeir curve may not satisfy the threshold tolerance limit. We compute the *squared distance* between each point p_k of digitized curve and its corresponding point $Q_k(t)$ of parametric curve.

$$\begin{aligned} d_k^2 &= |p_k - Q_k(t)| \\ &= [p_{x_k} - Q_{x_k}(t)]^2 + [p_{y_k} - Q_{y_k}(t)]^2 \end{aligned} \tag{4.13}$$

Among all the computed distances by (4.13) we find maximum squared distance d_{max}^2.

$$d_{max}^2 = Max(d_1^2, d_2^2, \ldots, d_n^2)$$

If d_{max}^2 exceeds predefined error tolerance limit d_{BP} then the segment is broken into two segments at the point of maximum distance and the point corresponding to maximum distance is added to list of significant points. Number of segments and number of Significant points are increased by one.

The process is repeated for each segment until all the segments of all the pieces meet the threshold tolerance limit. Figure (4.12) shows the flow chart of system.

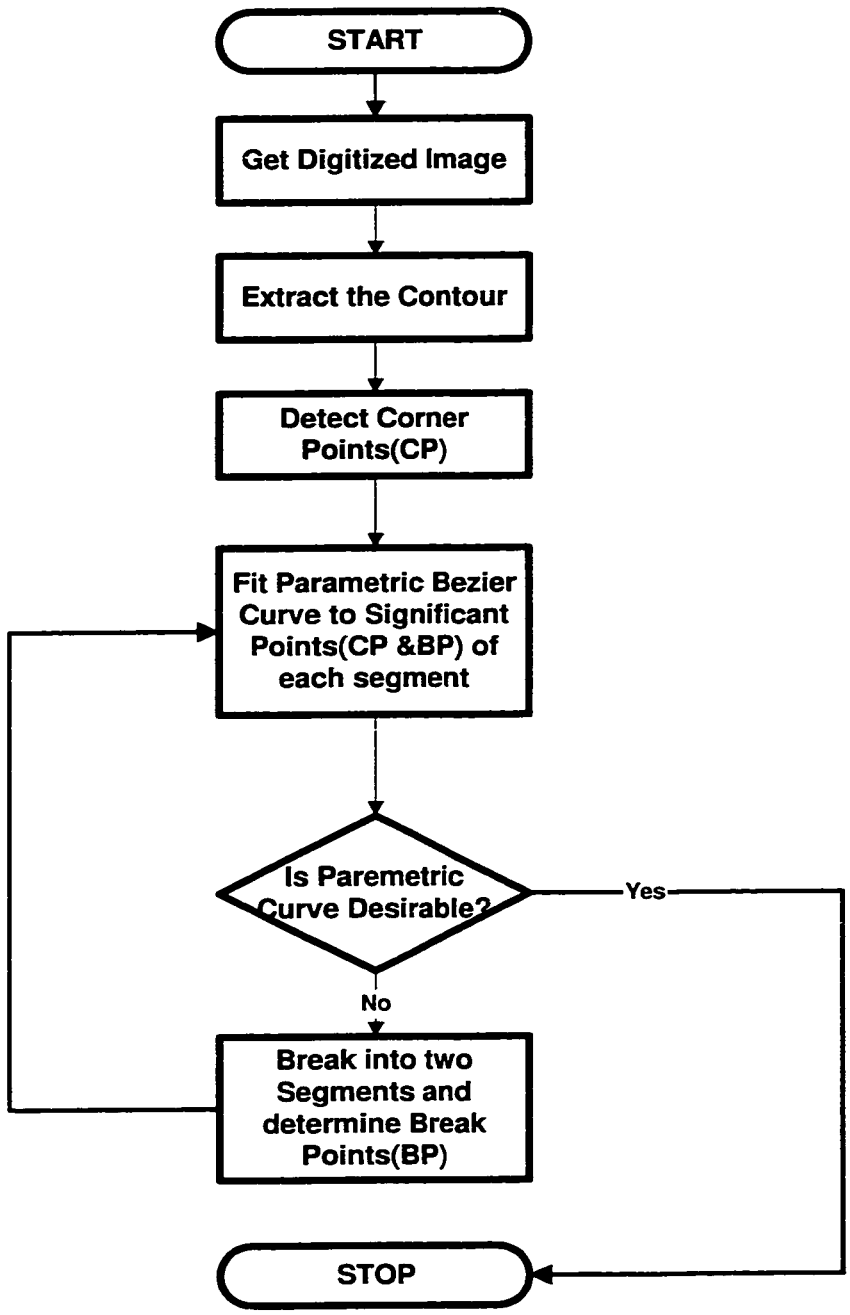


Figure 4.12: Font Design System with Filtering without Reprameterization

4.5 Filtering Noise

The points obtained by extracting the contour of digitized image may have noise (jagged edges). When filtering is needed to eliminate the jaggies is discussed by [21]. We can improve the performance of our system by removing the noise. So one more step of Filtering Noise is added to our system after corner detection step. Figure (4.13) shows the new flow chart of our system

There have been extensive work to filter/minimize the noise from sample data [12], [1]. To solve the problem we adopted a technique known as an approximation to a *Gaussian Filter*. Mathematically we can express *Gaussian Filter* as follows:

$$\begin{aligned} x_i &= 0.5x_i + 0.25x_{i+1} + 0.25x_{i-1} \\ y_i &= 0.5y_i + 0.25y_{i+1} + 0.25y_{i-1} \end{aligned} \tag{4.14}$$

This basic idea is to pass the sample data from the filter. The main task of filter is to remove the jagged effect in the sample data and hence to smoothen the outline by spreading out local variations. In our case each point is replaced by 1/2 weighted average of its own value and 1/4 weighted average of its immediate neighbor values. The filtering process is repeated number of times. By experimentation we found that six to ten iterations are enough to filter the noise from the contour. Figure (4.14) shows the digitized contour with noise and Figure (4.15) shows filtered contour upto six iterations. In Figure (4.15) effect of iteration on contour is shown from left to right and top to bottom.

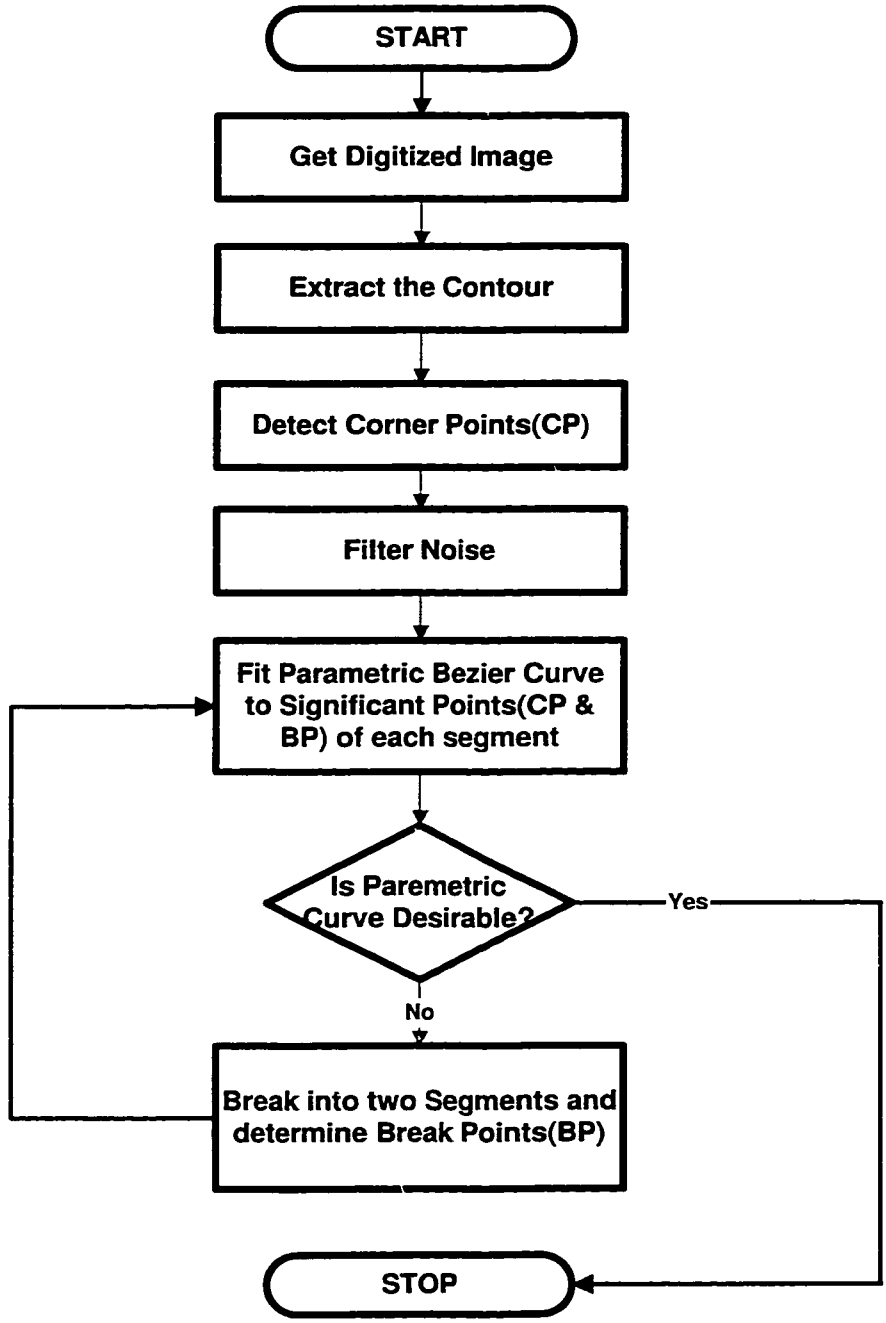


Figure 4.13: Font Design System with Filtering Process

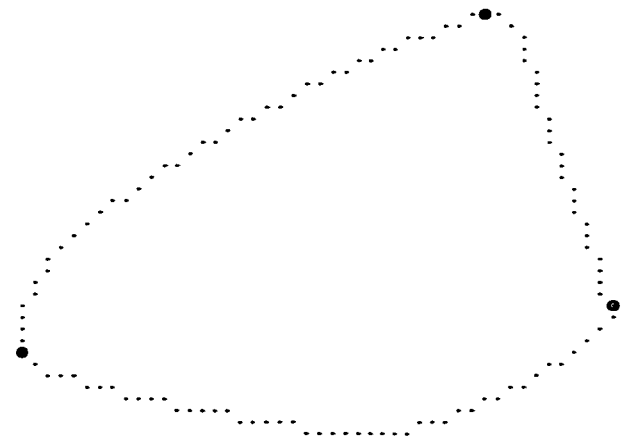


Figure 4.14: Digitized Contour with Noise

It is worth to mention that our Font Design System can work quite well without filtering. But definitely filtering improves the performance of the system by having need of lesser number of significant points. Figures of various characters are shown as examples by fitting Bezier to significant points (SP). Each example has two types of figures, first we have shown Bezier outline of the character without filtering contour data, then we have shown the same character produced by fitting Bezier to SP but noise is removed from contour. For character "Lillah" number of SP are 30 without filtering (Fig.4.16) and 25 with filtering (Fig.4.17). For character "Ali" number of SP are 36 without filtering (Fig.4.18) and 32 with filtering (Fig.4.19). For character "Samar" number of SP are 34 without filtering (Fig.4.20) and 31 with filtering (Fig.4.21).

Table (4.2) summaries the results.

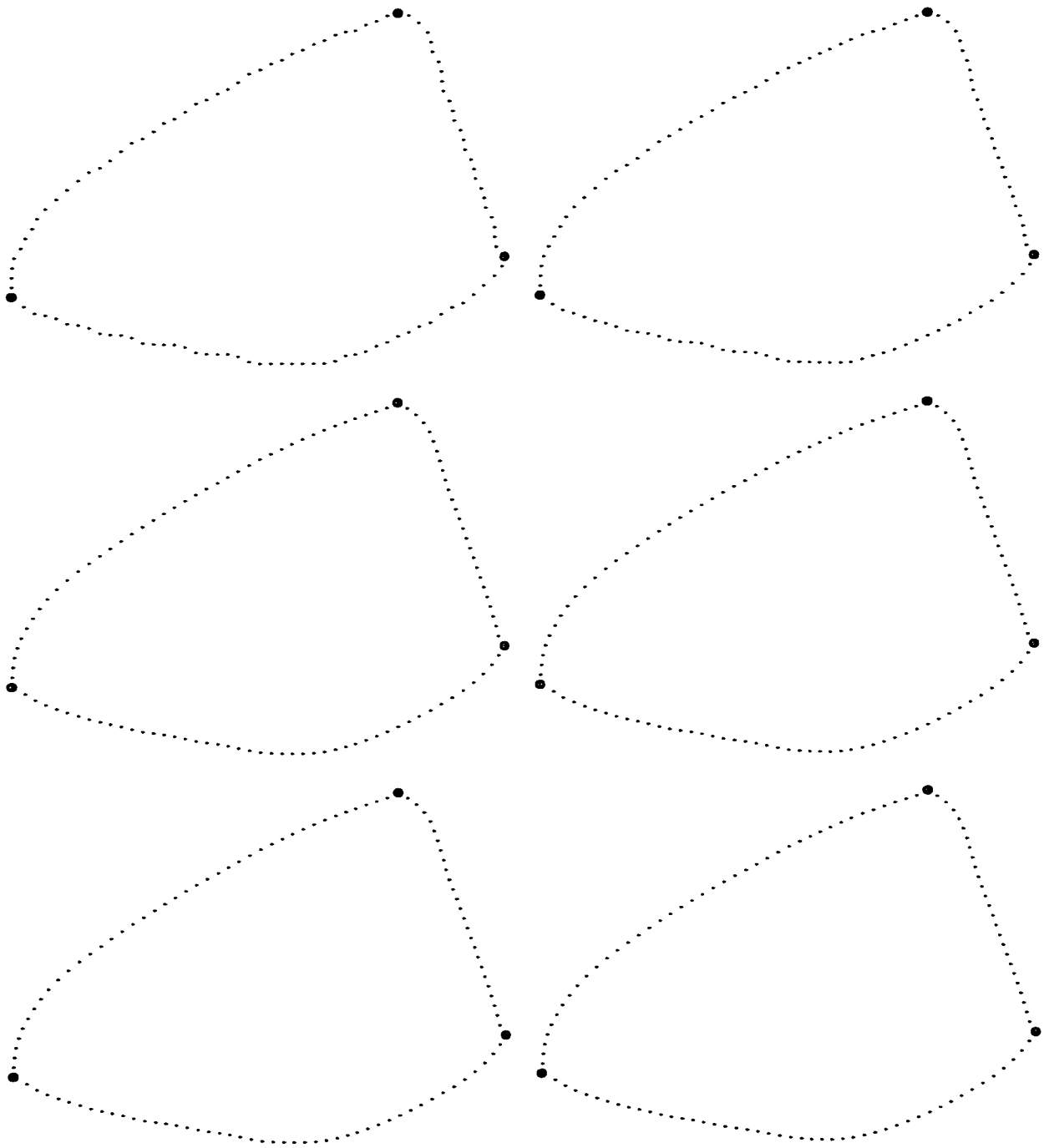


Figure 4.15: Removing Noise from Digitized Contour (6 iterations).

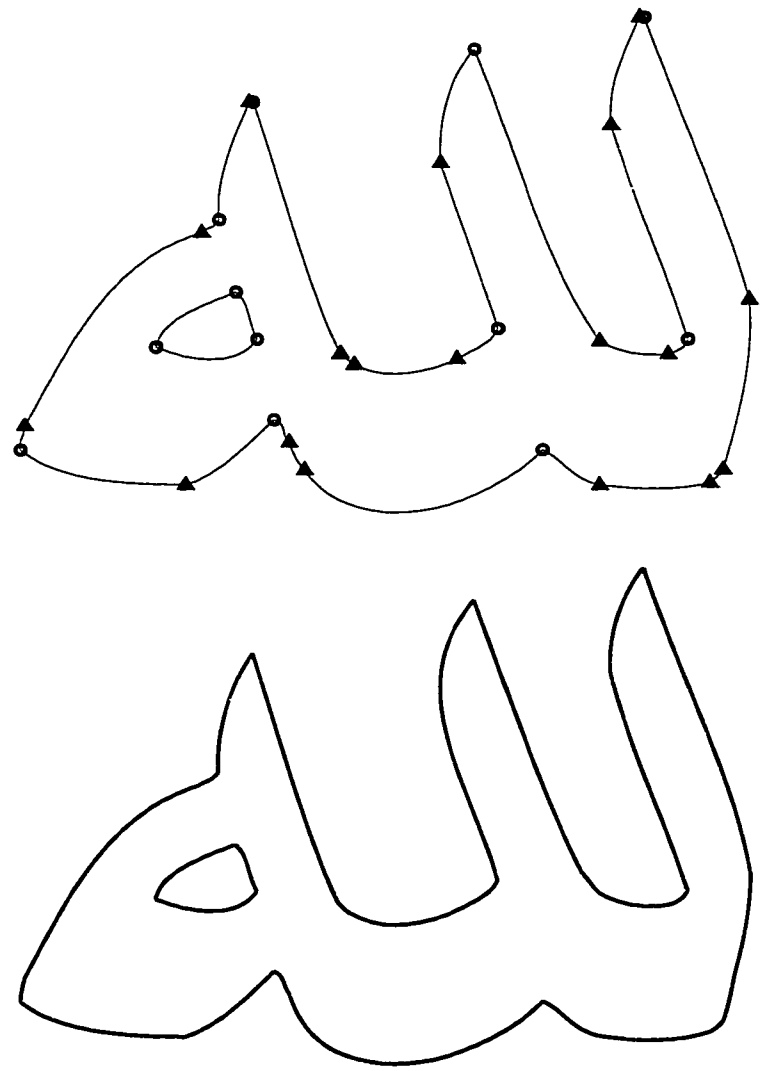


Figure 4.16: Fitted Bezier. Noise is not filtered from digitized Contour

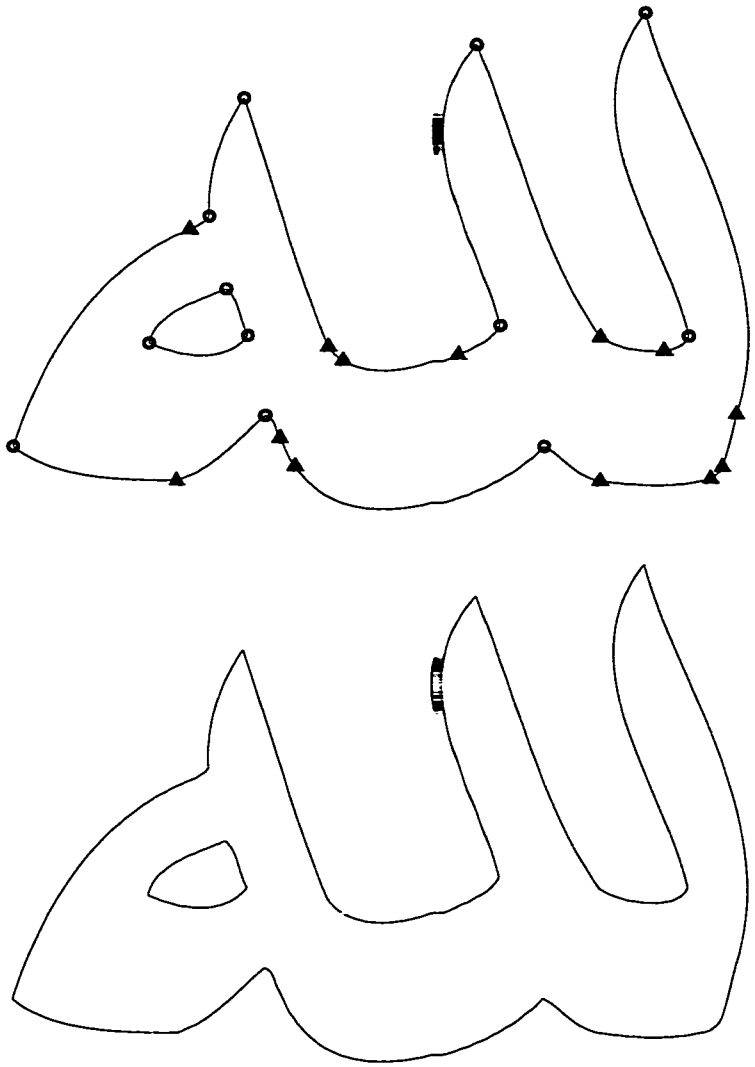


Figure 4.17: Fitted Bezier. Noise is filtered from digitized Contour

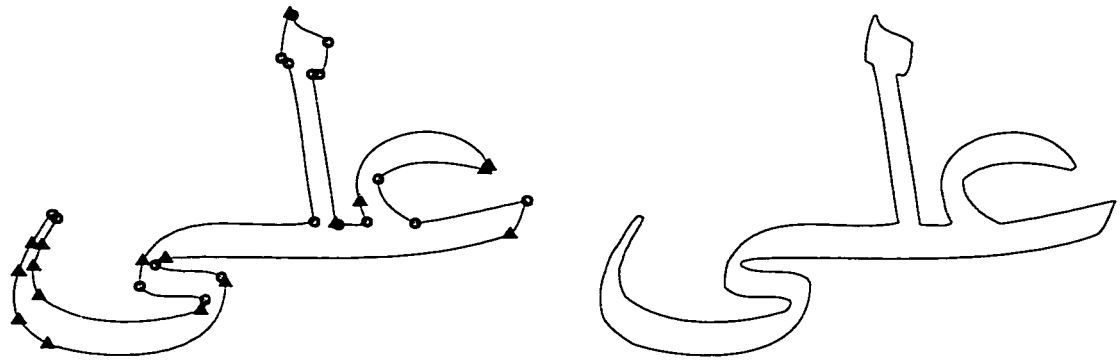


Figure 4.18: Fitted Bezier. Noise is not filtered from digitized Contour

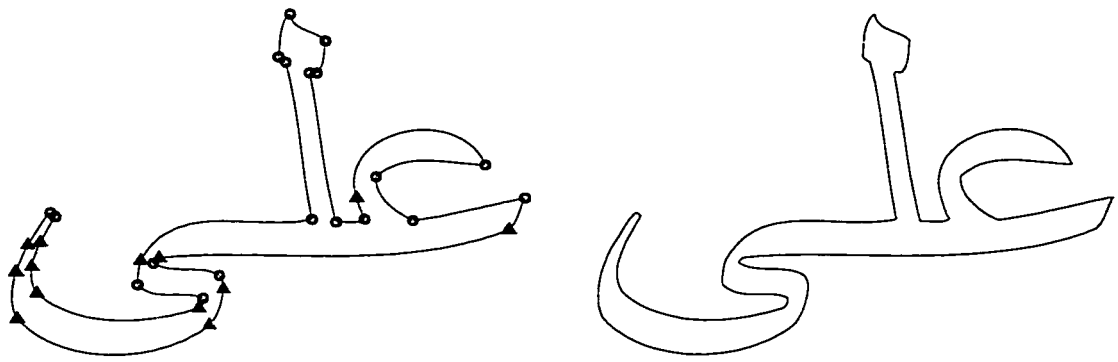


Figure 4.19: Fitted Bezier. Noise is filtered from digitized Contour.

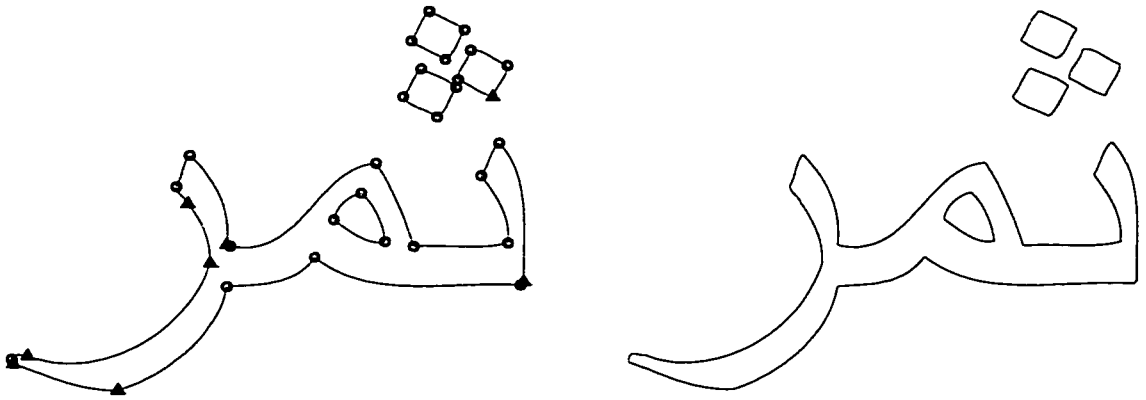


Figure 4.20: Fitted Bezier. Noise is not filtered from digitized Contour

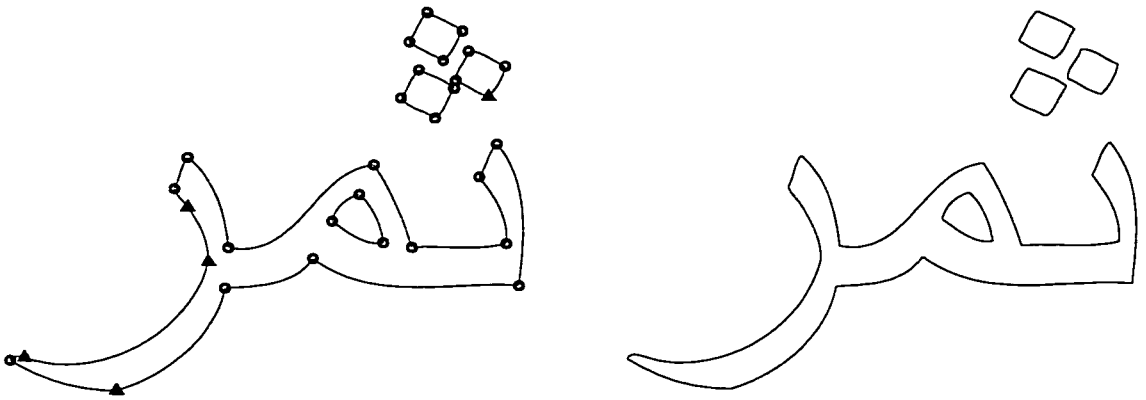


Figure 4.21: Fitted Bezier. Noise is filtered from digitized Contour.

Table 4.2: Effect of Noise Filtering (Six Iterations). $d_{BP} = 3.0$

# of Significant Points without filtering	# of Significant Points with filtering
36 Figure(4.18)	32 Figure(4.19)
30 Figure(4.16)	25 Figure (4.17)
34 Figure(4.20)	31 Figure (4.21)

Infact noise can be removed before detection of corners and then detecting corners from filtered data. But we find it more suitable to detect the corners first and then remove the noise from points other than corner points. Due to this strategy the overall shape of character is more intact. When corners are detected after removing noise the corner detection algorithm may not detect the same number of corners when they are detected before removing noise. For example the corner pointed to by an arrow in Figure (4.22) is missing in Figure (4.23).

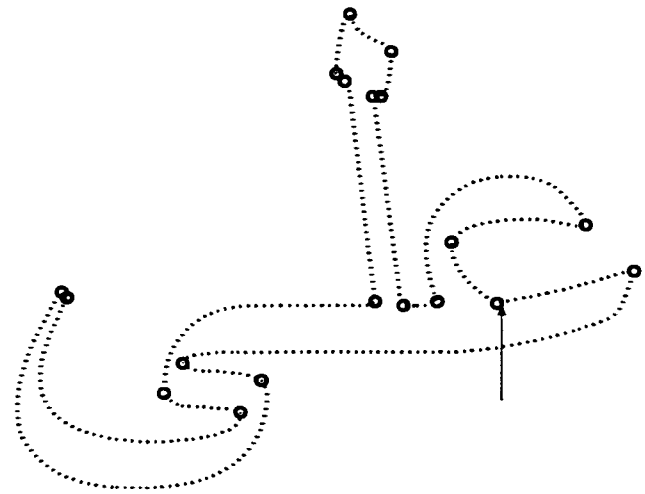


Figure 4.22: Filtered Boundary with Corners. Corners are detected before filtering.

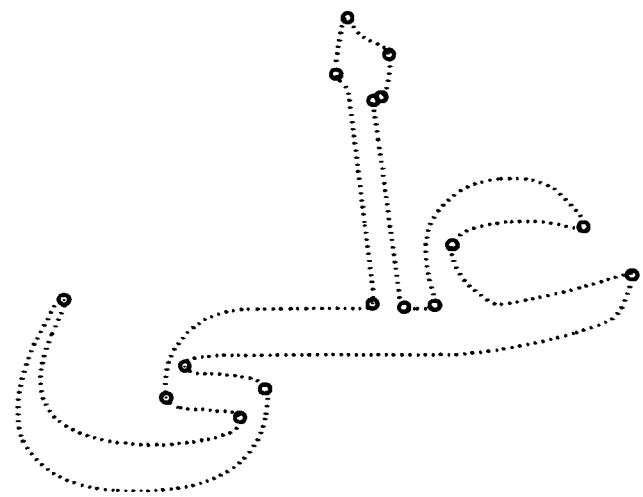


Figure 4.23: Filtered Boundary with Corners. Corners are detected after filtering.

4.6 Reparameterization

In this section we will describe how can we improve our fit by finding new values of t parameter. By having better values of t parameter we might not have to break a segment into one or more segments and hence we would need lesser number of Bezier curves. It means that we are going to introduce a new step in our algorithm.

We compare the original digitized curve and fitted parametric curve using squared distance criteria. If parametric curve does not meet the *threshold tolerance limit* we will apply the reparameterization step rather than breaking it into two segments. After reparameterization if the segment matches up to the desired tolerance limit then fine, otherwise we break them into two segments as previously we were doing. Figure (4.24) shows the modified flow chart of our system.

The reparameterization is explained as follows: Given a parametric curve $Q(t)$ and a point p, we have to find a point on the parametric curve closest to p, in other words, we have to find the parameter value t such that the distance from p to $Q(t)$ is minimum. This situation is illustrated in Figure (4.25). Note that distance from p to $Q(t)$ is perpendicular to the tangent i.e. $Q'(t)$ of the curve at $Q(t)$.

We can write the reparameterization equation as follows

$$[Q(t) - p].Q'(t) = 0 \tag{4.15}$$

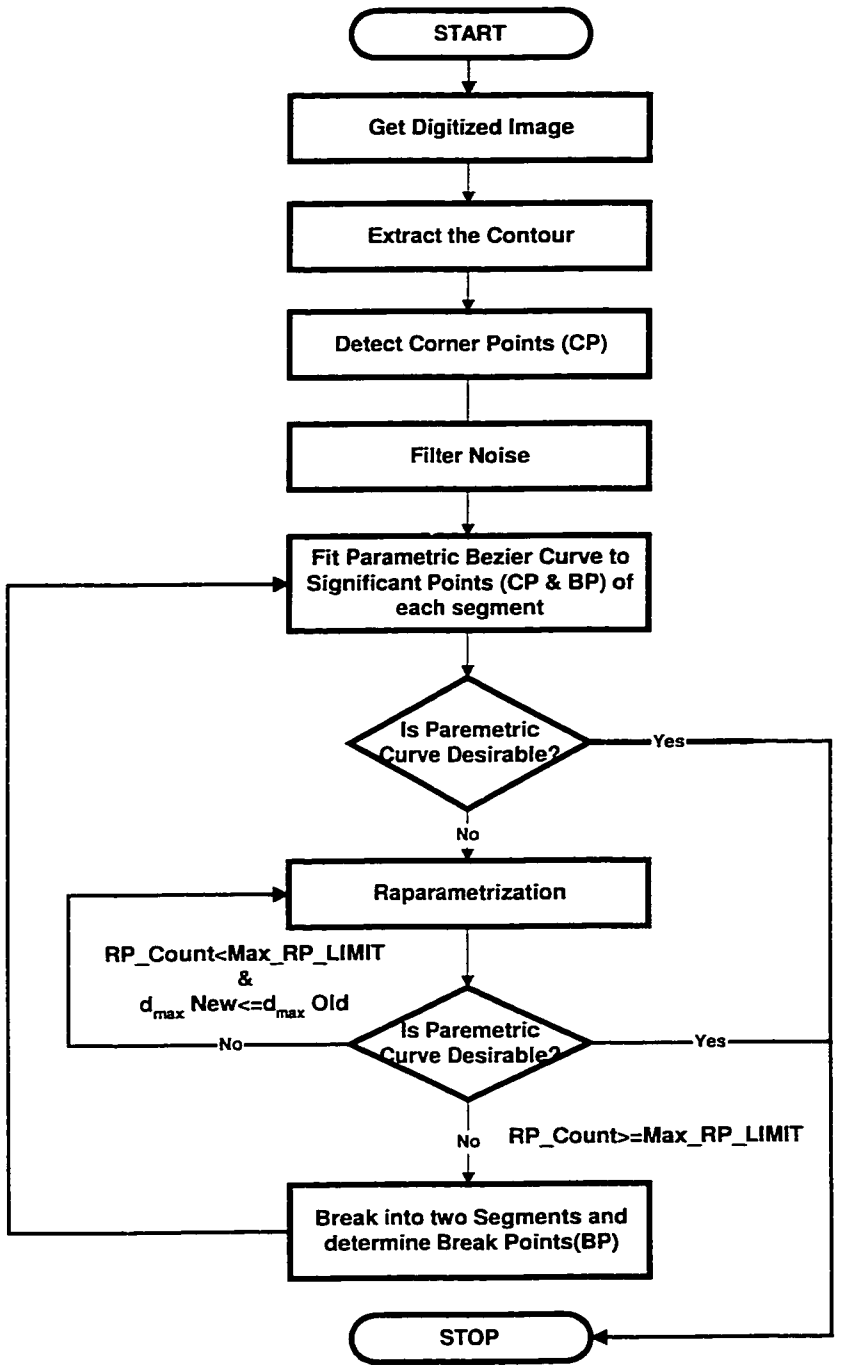


Figure 4.24: Font Design System with Filtering and Reparameterization

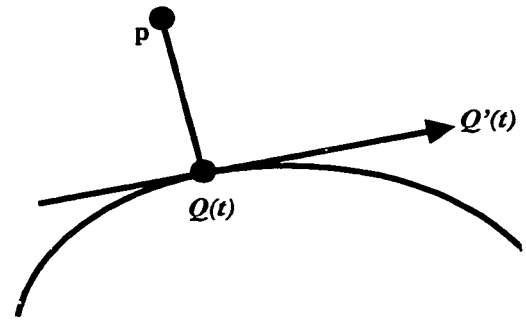


Figure 4.25: Distance between p and $Q(t)$

This is a quintic equation[1] in t and can be solved by Newton-Raphson's method. Another method of solving this equation is used in [29]. But this method is more expensive than Newton-Raphson's method. So we will give details of the later.

$[Q(t) - p]$ is polynomial of degree three and $Q'(t)$ is of degree two. So equation (4.15) is polynomial of degree five. Let

$$Q_1(t) = Q(t) - p \tag{4.16}$$

$$Q_2(t) = Q'(t) \tag{4.17}$$

Now we can write equation (4.15) as follows

$$Q_1(t).Q_2(t) = 0 \tag{4.18}$$

[1]There is dot product of $[Q(t) - p]$ and $Q'(t)$.

Writing equation (4.16) in Cartesian form

$$Q_1(t) = [Q_x(t) - p_x]i + [Q_y(t) - p_y]j \tag{4.19}$$

Let

$$Q_{1x}(t) = Q_x(t) - p_x \tag{4.20}$$

$$Q_{1y}(t) = Q_y(t) - p_y \tag{4.21}$$

The we can write equation (4.19) as follows

$$Q_1(t) = Q_{1x}(t)i + Q_{1y}(t)j \tag{4.22}$$

Writing equation (4.17) in Cartesian form

$$Q_2(t) = Q'_x(t)i + Q'_y(t)j \tag{4.23}$$

where

$$Q'_x(t) = 3[(1-t)^2(P_{x_1} - P_{x_0}) + 2t(1-t)(P_{x_2} - P_{x_1}) + t^2(P_{x_3} - P_{x_2})] \tag{4.24}$$

$$Q'_y(t) = 3[(1-t)^2(P_{y_1} - P_{y_0}) + 2t(1-t)(P_{y_2} - P_{y_1}) + t^2(P_{y_3} - P_{y_2})] \tag{4.25}$$

4.6.1 Newton-Raphson Method

We already have initial approximation of roots (i.e. t parameter). The formula for Newton-Raphson method to improve the roots is following.

$$t \leftarrow t - \frac{f(t)}{f'(t)} \tag{4.26}$$

In our case

$$\begin{aligned} f(t) &= Q_1(t).Q_2(t) \\ &= [Q_{1x}(t)i + Q_{1y}(t)j].[Q'_x(t)i + Q'_y(t)j] \end{aligned} \tag{4.27}$$

Dot product yields the following result

$$f(t) = Q_{1x}(t)Q'_x(t) + Q_{1y}(t)Q'_y(t) \tag{4.28}$$

Let

$$f_x(t) = Q_{1x}(t)Q'_x(t) \tag{4.29}$$

$$f_y(t) = Q_{1y}(t)Q'_y(t) \tag{4.30}$$

Then

$$f(t) = f_x(t) + f_y(t) \tag{4.31}$$

We now solve the equation for $f_x(t)$. The equation for $f_y(t)$ can be obtained through symmetry.

$$\begin{aligned} f_x(t) = {} & [(1-t^3)P_{x_0} + 3t(1-t^2)P_{x_1} + 3t^2(1-t)P_{x_2} + t^3P_{x_3} - p_x] \\ & 3[(1-t)^2(P_{x_1} - P_{x_0}) + 2t(1-t)(P_{x_2} - P_{x_1}) + t^2(P_{x_3} - P_{x_2})] \end{aligned} \tag{4.32}$$

After simplification we can write $f_x(t)$ as follows

$$f_x(t) = 3[f_{x_1}(t) + f_{x_2}(t) + f_{x_3}(t) + f_{x_4}(t) - f_{x_5}(t)] \tag{4.33}$$

where

$$
\begin{aligned}
f_{x_1}(t) &= (1-t)^5 P_{x_0}(P_{x_1} - P_{x_0}) + 2t(1-t)^4 P_{x_0}(P_{x_2} - P_{x_1}) + t^2(1-t)^3 P_{x_0}(P_{x_3} - P_{x_2}) \\
f_{x_2}(t) &= 3t(1-t)^4 P_{x_1}(P_{x_1} - P_{x_0}) + 6t^2(1-t)^3 P_{x_1}(P_{x_2} - P_{x_1}) + 3t^2(1-t)^2 P_{x_1}(P_{x_3} - P_{x_2}) \\
f_{x_3}(t) &= 3t^2(1-t)^3 P_{x_2}(P_{x_1} - P_{x_0}) + 6t^3(1-t)^2 P_{x_2}(P_{x_2} - P_{x_1}) + 3t^4(1-t) P_{x_2}(P_{x_3} - P_{x_2}) \\
f_{x_4}(t) &= t^3(1-t)^2 P_{x_3}(P_{x_1} - P_{x_0}) + 2t^4(1-t) P_{x_3}(P_{x_2} - P_{x_1}) + t^5 P_{x_3}(P_{x_3} - P_{x_2}) \\
f_{x_5}(t) &= (1-t)^2(P_{x_1} - P_{x_0})p_x + 2t(1-t)(P_{x_2} - P_{x_1})p_x + t^2(P_{x_3} - P_{x_2})p_x
\end{aligned}
$$

Similarly equation for $f_y(t)$ can be written.

$$f_y(t) = 3[f_{y_1}(t) + f_{y_2}(t) + f_{y_3}(t) + f_{y_4}(t) - f_{y_5}(t)] \tag{4.34}$$

Now for derivative of $f(t)$ i.e. $f'(t)$

$$f'(t) = f'_x(t) + f'_y(t) \tag{4.35}$$

where

$$
\begin{aligned}
f'_x(t) &= 3[f'_{x_1}(t) + f'_{x_2}(t) + f'_{x_3}(t) + f'_{x_4}(t) - f'_{x_5}(t)] \\
f'_y(t) &= 3[f'_{y_1}(t) + f'_{y_2}(t) + f'_{y_3}(t) + f'_{y_4}(t) - f'_{y_5}(t)]
\end{aligned} \tag{4.36}
$$

Finding $f'_{x_1}(t), f'_{x_2}(t), f'_{x_3}(t), f'_{x_4}(t)$ and $f'_{x_5}(t)$

$$
\begin{aligned}
f'_{x_1}(t) =& P_{x_0}[(1-t)^4\{-5(P_{x_1}-P_{x_0})+2(P_{x_2}-P_{x_1})\} \\
&+ t(1-t)^3\{-8(P_{x_2}-P_{x_1})+2(P_{x_3}-P_{x_2})\} \\
&- 3t^2(1-t)^2\{(P_{x_3}-P_{x_2})\}]
\end{aligned} \tag{4.37}
$$

$$
\begin{aligned}
f'_{x_2}(t) =& 3P_{x_1}[(1-t)^4\{(P_{x_1}-P_{x_0})\} \\
&+ t(1-t)^3\{-4(P_{x_1}-P_{x_0})+4(P_{x_2}-P_{x_1})\} \\
&+ t^2(1-t)^2\{-6(P_{x_2}-P_{x_1})+3(P_{x_3}-P_{x_2})\} \\
&+ t^3(1-t)\{-2(P_{x_3}-P_{x_2})\}]
\end{aligned} \tag{4.38}
$$

$$
\begin{aligned}
f'_{x_3}(t) =& 3P_{x_2}[t(1-t)^3\{2(P_{x_1}-P_{x_0})\} \\
&+ t^2(1-t)^2\{-3(P_{x_1}-P_{x_0})+6(P_{x_2}-P_{x_1})\} \\
&+ t^3(1-t)\{-4(P_{x_2}-P_{x_1})+4(P_{x_3}-P_{x_2})\} \\
&+ t^4\{-(P_{x_3}-P_{x_2})\}]
\end{aligned} \tag{4.39}
$$

$$
\begin{aligned}
f'_{x_4}(t) =& 3P_{x_3}[t^2(1-t)^2\{3(P_{x_1}-P_{x_0})\} \\
&+ t^3(1-t)\{-2(P_{x_1}-P_{x_0})+8(P_{x_2}-P_{x_1})\} \\
&+ t^4\{-2(P_{x_2}-P_{x_1})+5(P_{x_3}-P_{x_2})\}]
\end{aligned}
\tag{4.40}
$$

$$
\begin{aligned}
f'_{x_5}(t) =& p_x[(1-t)\{-2(P_{x_1}-P_{x_0})+2(P_{x_2}-P_{x_1})\} \\
&+ t\{-2(P_{x_2}-P_{x_1})+2(P_{x_3}-P_{x_2})\}]
\end{aligned}
\tag{4.41}
$$

Similar set of derivatives can be written for $f'_{y_1}(t), f'_{y_2}(t), f'_{y_3}(t), f'_{y_4}(t)$ and $f'_{y_5}(t)$.

Now we have all the necessary information needed to solve equation (4.26) and we can find new value of t. Using these new values of parameter t we find new control points and apply the fitting process as usual. We do reparameterization of the segments not fulfilling the threshold tolerance limit. But as we have described before that reparameterization is expensive process, therefore we fix a maximum limit on number of times a segment can go reparameterizaton. If either the maximum limit of reparameterization exceeds or reparameterization increases the square distance between digitized curve and parametric curve rather than decreasing it (in some cases it is possible) then we break the segment into two segments at point of maximum distance. The effect of reparameterization is shown in Figure (4.26),

Figure (4.27) and Figure (4.28).

When all the segments meet the square distance threshold limit then there is no need to keep the specific t values. We used specific t values to find the best possible intermediate control points and providing initial estimate for finding new t values in Newton-Raphson' method. After having all the control points for all the segments there is no more need of specific t values. We can write a general expression for finding t values. By this general expression we can find t values on the fly (i.e. during fitting Bezeir to a segment). If we fit a cubic Bezier to a segment using n points then the expression of t can be written as follows.

$$t_i = \begin{cases} 0 & \text{if} \quad i = 1; \\ t_{i-1} + 1/(n-1) & \text{if} \quad 2 \leq i \leq n-1; \\ 1 & \text{if} \quad i = n \end{cases}$$

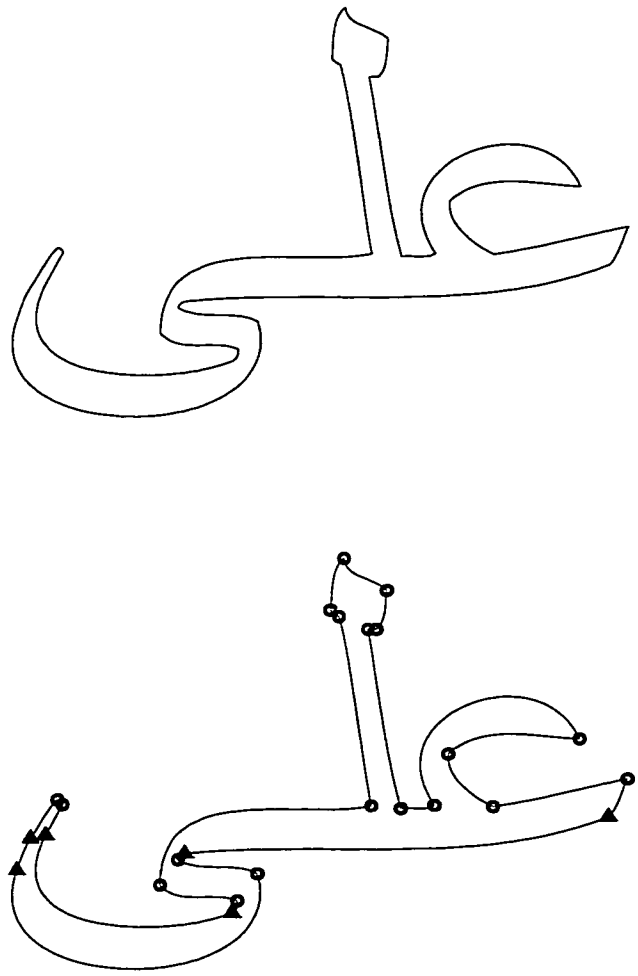


Figure 4.26: Bezier Outline with Filtering and Reparameterization

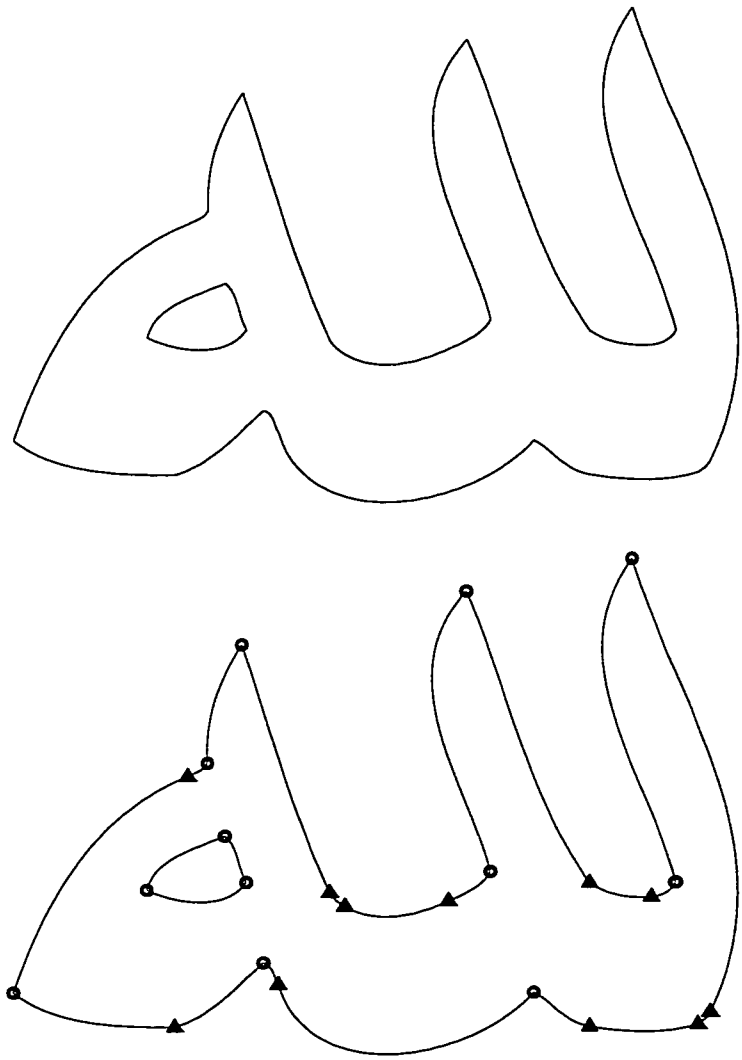


Figure 4.27: Bezier Outline with Filtering and Reparameterization

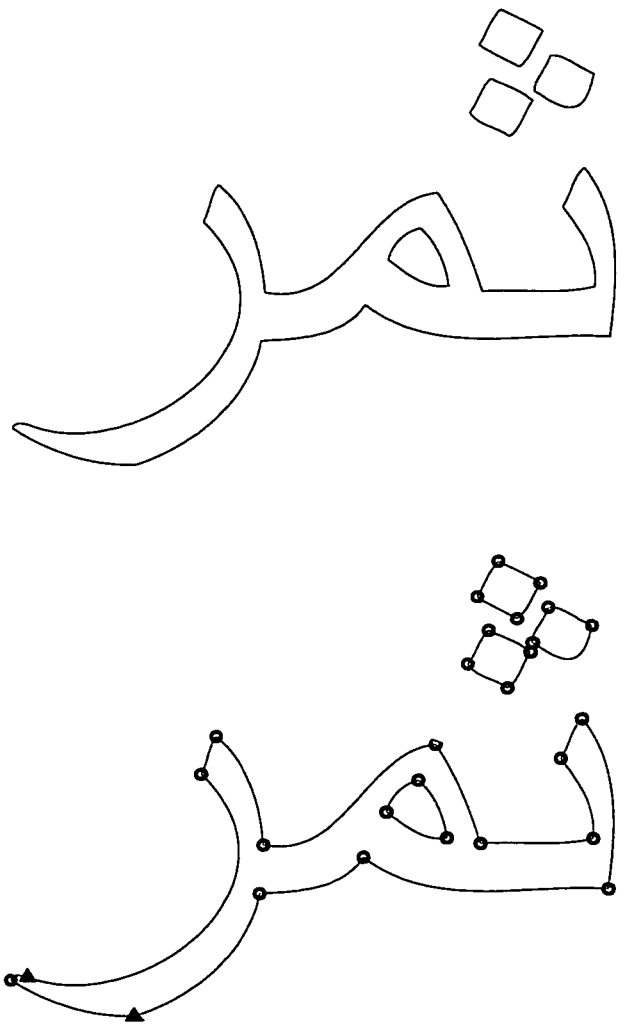


Figure 4.28: Bezier Outline with Filtering and Reparameterization

4.7 Conclusion

In this chapter we described details of our Font Design System. Initially we described the basic steps of our system then we described those steps which can improve the performance of the system such as Filtering Noise and Reparameterization steps. The curve fitting step is the main step of our system. We used cubic Bezier to fit the curve. The two corner points for each segment are taken as the two end control points of Bezier curve whereas the two intermediate control points of Bezier curve are determined by solving the sum of squared distances equation. Table (4.3) summaries our results.

Table 4.3: Effect of Noise Filtering (Six Iterations) and Reparameterizaton (Eleven Iterations). $d_{BP} = 3.0$

# of Significant Points without Reparameterization (Contour is not Filtered)	# of Significant Points without Reparameterization (Contour is Filtered)	# of Significant Points with Reparameterization (Contour is Filtered)
36 Figure(4.18)	32 Figure(4.19)	25 Figure (4.26)
30 Figure(4.16)	25 Figure (4.17)	23 Figure (4.27)
34 Figure(4.20)	31 Figure (4.21)	27 Figure (4.28)

Chapter 5

EVALUATION AND ANALYSIS

In this chapter we will evaluate and analyze our font design system.

5.1 Data of Outline Fonts

In chapter 1 we mentioned several advantages of outline representation of fonts as compared to bitmap representation. One advantage is less storage space is required to store outline fonts. We will explain this with one example. Figure (5.1) shows a parametric representation of "lillah" character. To draw this character we need to store the information shown in Figure (5.2) and (5.3). (Note that only numeric values need to be stored). A comparison in terms of size is given in Table (5.1). We can further reduce our data set by noting that last control point (i.e. P_3) of each segment is the first control point (i.e. P_0) of next segment and first control point of

first segment and last control point of last segment are also equal (because we are dealing with closed boundary curves).

If very large set of characters/Fonts (alphabets/symbols/numeric,punctuation marks etc.) is to be designed then data set can be reduced by classifying characters into groups. For example in Arabic language the shapes of "eem", "Ha" and "Kha" are similar, they can be placed in a same group. Many segments of these characters are same. For similar segments only one set of data is needed to be save rather than storing separate data set of each character. A scheme for compacting fonts is proposed in [31].

The developed outline fonts need to be rasterize (i.e. converting to bitmap and filling) just before display or printing. Rasterization topic is beyond the scope of our study. Several standard algorithms for rasterization can be found in literature [8], [19], [18].

Table 5.1: Comparison in term of size

Size of Bitmap file (in bytes)	Size of Boundary data file (in bytes)	Size of Parametric data file (in bytes)
14270	19148	1045

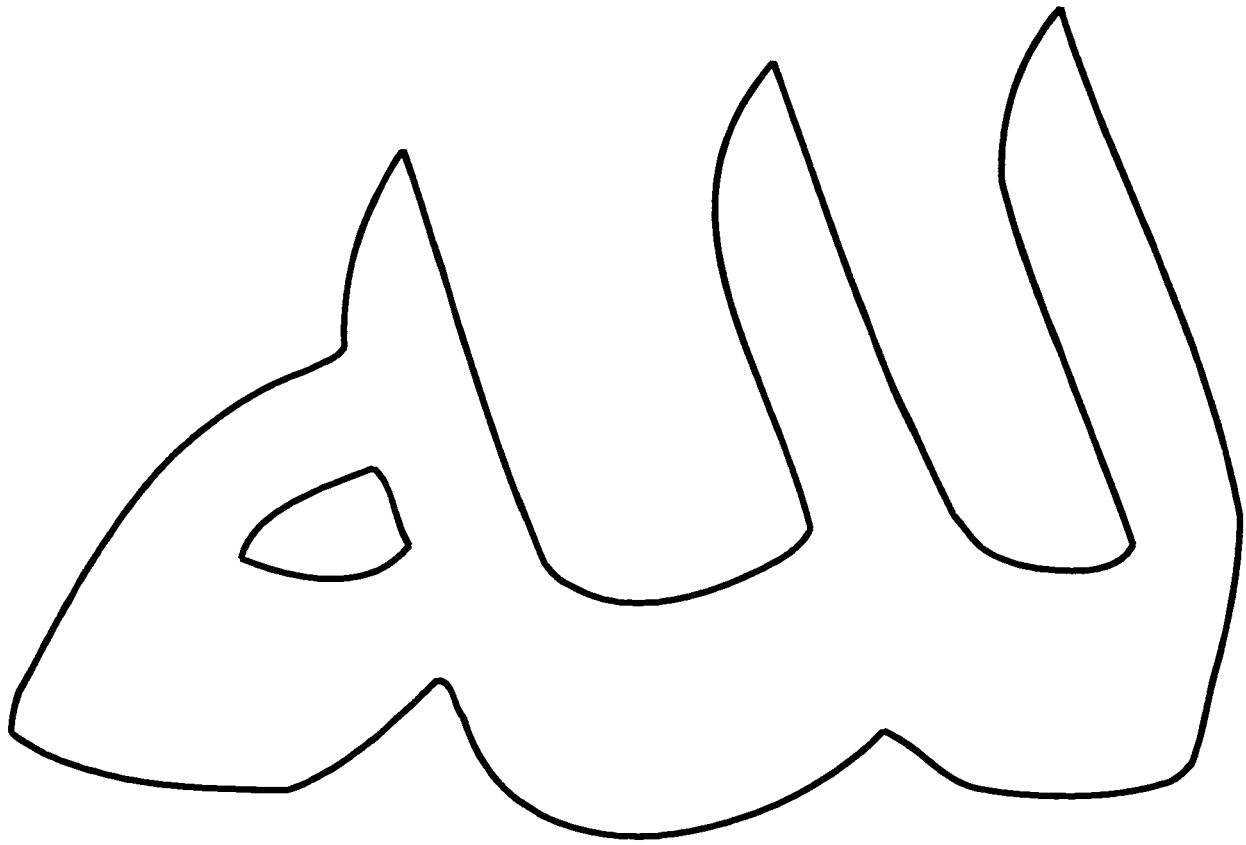


Figure 5.1: Parametric Representation of lillah character

Number of Pieces	**2**
Number of Boundary Points in First Piece	**1522**
Number of Boundary Points in Second Piece	**116**
Number of Segments in First Piece	**25**
Number of Segments in Second Piece	**3**

Number of points in each Segments of First Piece	*Number of points in each Segments of Second Piece*
15	27
43	43
77	49
14	
115	
11	
66	
134	
9	
48	
22	
153	
147	
12	
30	
14	
118	
59	
165	
52	
29	
8	
51	
28	
127	

Figure 5.2: Parametric Data for 'lillah' character (cont.)

Control Points for First Piece

P_{0x}	P_{0y}	P_{1x}	P_{1y}	P_{2x}	P_{2y}	P_{3x}	P_{3y}
128	62	125	66	126	72	121	74
121	74	109	61	97	46	80	39
80	39	57	39	22	41	5	58
5	58	5	62	6	67	7	71
7	71	25	110	43	157	86	174
86	174	89	176	93	177	95	181
95	181	94	203	100	226	111	244
111	244	124	200	134	154	150	111
150	111	152	108	154	105	157	104
157	104	171	94	190	99	204	106
204	106	211	110	220	114	223	122
223	122	212	172	176	224	212	272
212	272	228	223	241	173	261	126
261	126	264	122	266	118	269	115
269	115	277	109	288	107	298	108
298	108	303	109	307	111	309	116
309	116	298	155	283	193	274	233
274	233	273	252	278	273	289	289
289	289	304	234	329	181	338	125
338	125	338	108	335	91	331	74
331	74	329	65	328	55	325	46
325	46	323	43	321	41	319	40
319	40	303	34	285	35	269	38
269	38	258	40	252	52	243	57
243	57	214	21	143	3	128	62

Control Points for Second Piece

P_{0x}	P_{0y}	P_{1x}	P_{1y}	P_{2x}	P_{2y}	P_{3x}	P_{3y}
113	117	109	125	109	136	103	142
103	142	90	136	70	129	67	113
67	113	81	106	103	101	113	117

Figure 5.3: Parametric Data for 'lillah' character

5.2 Comparison with Koichi Itoh Method

A method for capturing outline font is also proposed by Itoh [17] but there are many differences between our method and his method. The mathematical form he used to represent polynomial is

$$\begin{aligned} B_x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ B_y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \end{aligned} \tag{5.1}$$

The mathematical form we used to represent polynomial is

$$\begin{aligned} Q_x(t) &= (1-t^3)P_{x_0} + 3t(1-t^2)P_{x_1} + 3t^2(1-t)P_{x_2} + t^3 P_{x_3} \\ Q_y(t) &= (1-t^3)P_{y_0} + 3t(1-t^2)P_{y_1} + 3t^2(1-t)P_{y_2} + t^3 P_{y_3} \end{aligned} \tag{5.2}$$

Equation (5.1) requires to find four coefficients a, b, c and d. In our method we find only two intermediate control points i.e. P_1 and P_2 of cubic Bezier, because at $t = 0$ and $t = 1$, the equation (5.2) will reduce to first and last control points i.e. P_0 and P_3 respectively. Due to this the parametric curve of Itoh's method does not guarantee to pass through corner points. Consequently, segments does not intersect with each other. Therefore Itho used Bezier Clipping [30] to find the intersection of segments. A method of finding intersection of Bezier curves is also proposed in [5]. Because in Itho's method corner points are not preserved therefore we feel that it will effect the accuracy of the system. Figure (5.4) shows the outline produced by

Itho's method prior to finding intersection of segments.

We also used Noise filtering to improve the performance of our system but Itoh [17] did not mention any kind of filtering in his paper.

5.3 An Alternate Approach

Zhang [37],[38] described a method of vectorization of Digital Images. He showed application of his method by fitting curves to cartoons images but the method can be used for font design as well . Important thing about his method is that he used Recursive Algebraic technique for curve fitting. This is an alternative of Parametric representation of curves. In his method Zhang used quadratic algebraic curve segments to vectorize the digital images. This requires Traingulation of the digital curve, a procedure for devising a triangle to contain the digital curve so that one or more pieces of quadratic algebraic curves can be found to approximate it. Subdivision is used to achieve the closeness between original digital curve and algebraic one. See [9] for some recent work about computing triangle strips.

But as we have described in chapter 2 that this kind of curve representation has certain problems. For example the modeling capability of algebraic curve is limited, the algebraic equation may have more than one solution, so constraints are required to overcome this problem. Also at join point it is difficult to determine whether their tangent directions agree or not.

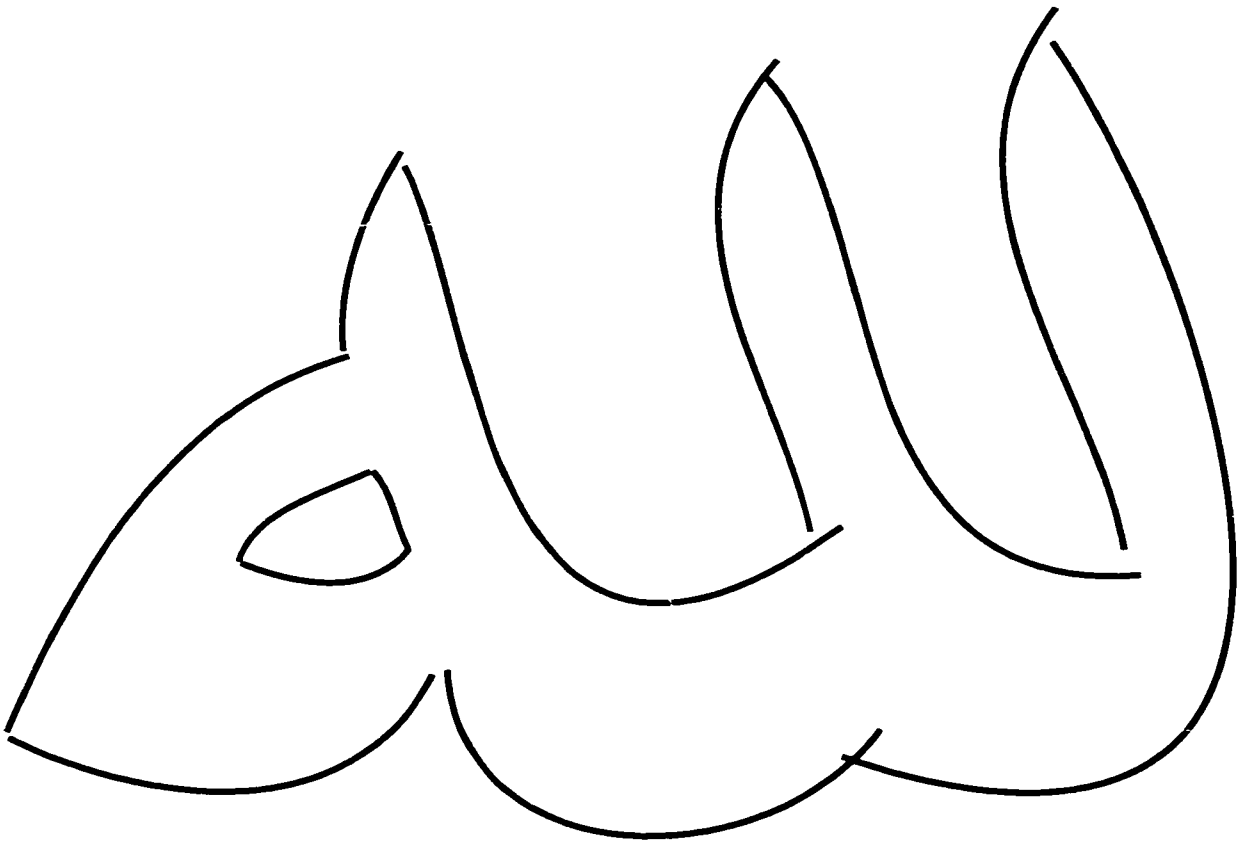


Figure 5.4: Parametric Representation of lillah character by Itho's metohd

Chapter 6

Transformations and Mapping

This chapter deals with producing various sizes and shapes of fonts using Transformations and Mapping. As we described in chapter 1 that outline fonts have many advantages over bitmap fonts. By manipulation of small number of points, various sizes and shapes of the same character can obtained very efficiently.

In chapter 2, we discussed some of the properties of Bezier Curves. One property of Bezier Curves is that they are *affine invariant.* It means that we can rotate, translate, or scale the set of control points, and the curve associated with them will retain its relationship with the control points.

Section §6.1 describes Transformations while §6.2 is related to Mapping.

6.1 Transformations

6.1.1 Scaling

A Point $P = (x, y)$ can be *scaled* along x-axis and/or y-axis. If s_x and s_y are amounts of scaling along x-axis and y-axis respectively then mathematically we can write

$$\begin{aligned} x' &= x.s_x \\ y' &= y.s_y \end{aligned} \tag{6.1}$$

where x' and y' are new values of x and y i.e. $P' = (x', y')$.

In matrix notation the above equation can be expressed as follows

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} \tag{6.2}$$

If we do not want to scale along an axis then its scaling factor (s_x or s_y) is set to 1. Figure (6.1) shows the character "Allah" before scaling. Figure (6.2) shows scaling along x-axis ($s_x = 1/2$, $s_y = 1$), Figure (6.4) shows scaling along y-axis ($s_x = 1$, $s_y = 1/2$) and Figure (6.6) shows scaling along both x-axis and y-axis ($s_x = 1/2$, $s_y = 1/2$).

It should be noted that to do scaling we multiply scaling factors (i.e. s_x and s_y) with only control points and then fit the Cubic Bezier to new set if points.

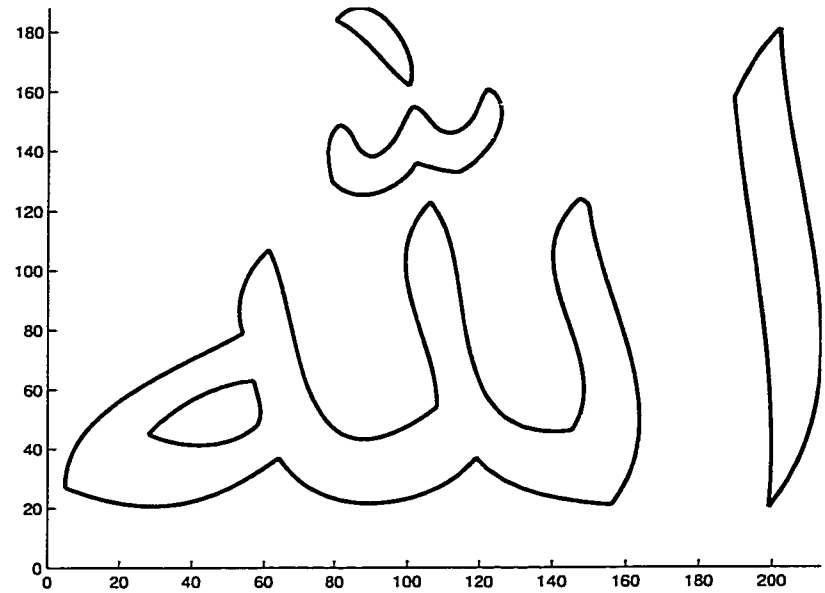


Figure 6.1: Before Scaling

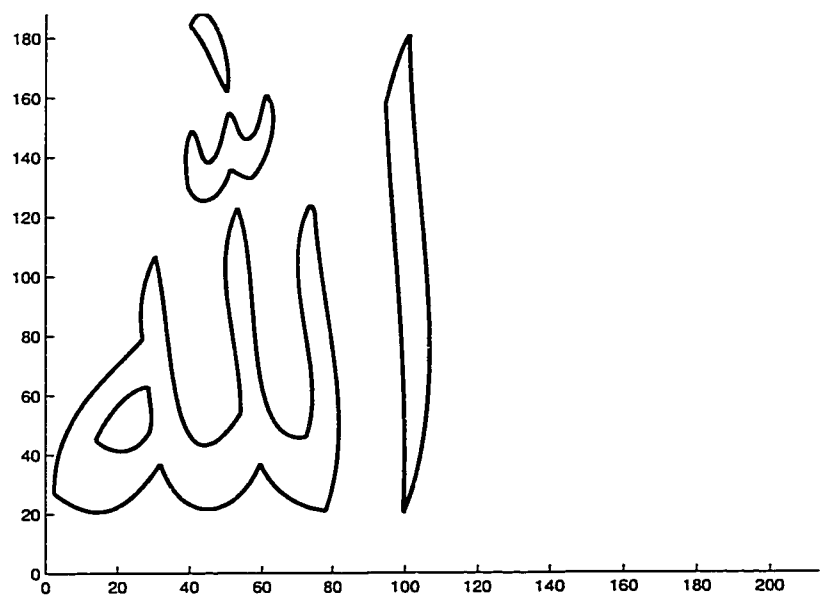


Figure 6.2: Scaling along x-axis. $s_x = 1/2, s_y = 1$

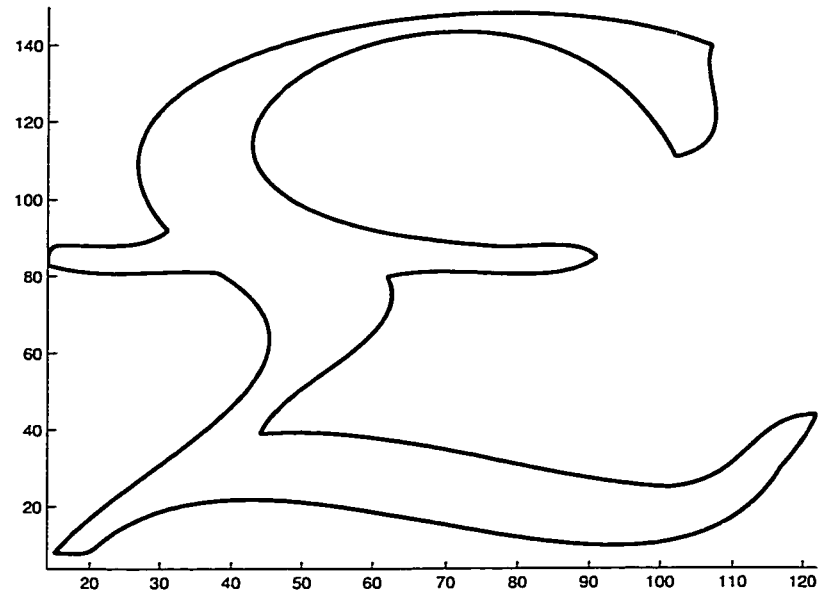


Figure 6.3: Before Scaling

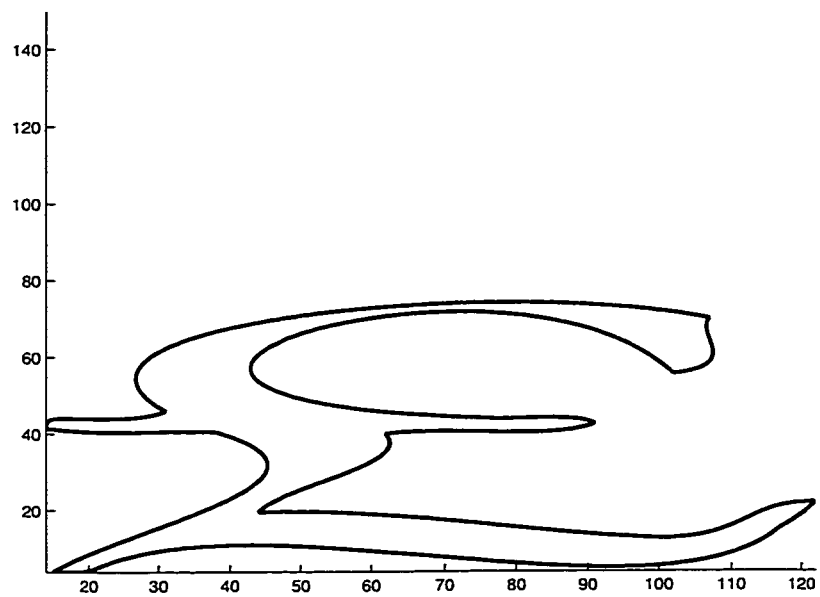


Figure 6.4: Scaling along y-axis. $s_x = 1, s_y = 1/2$

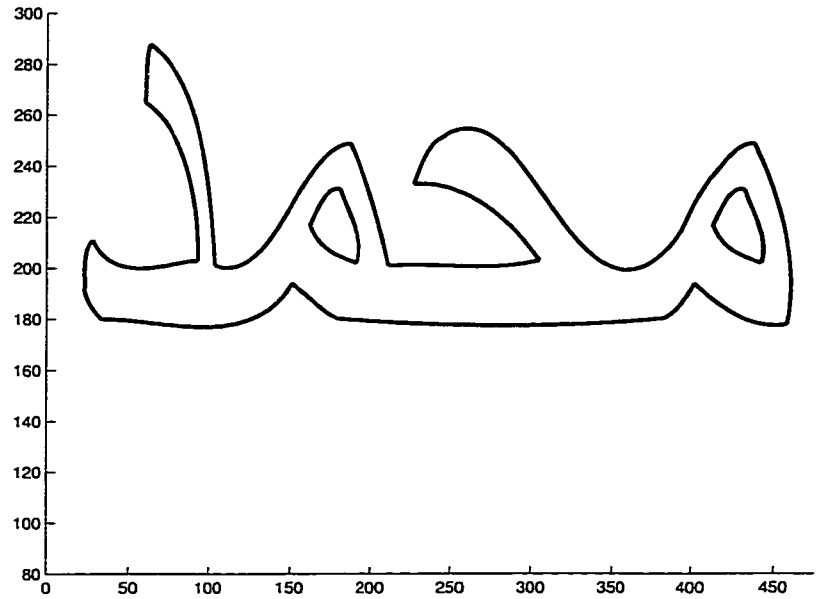


Figure 6.5: Before Scaling

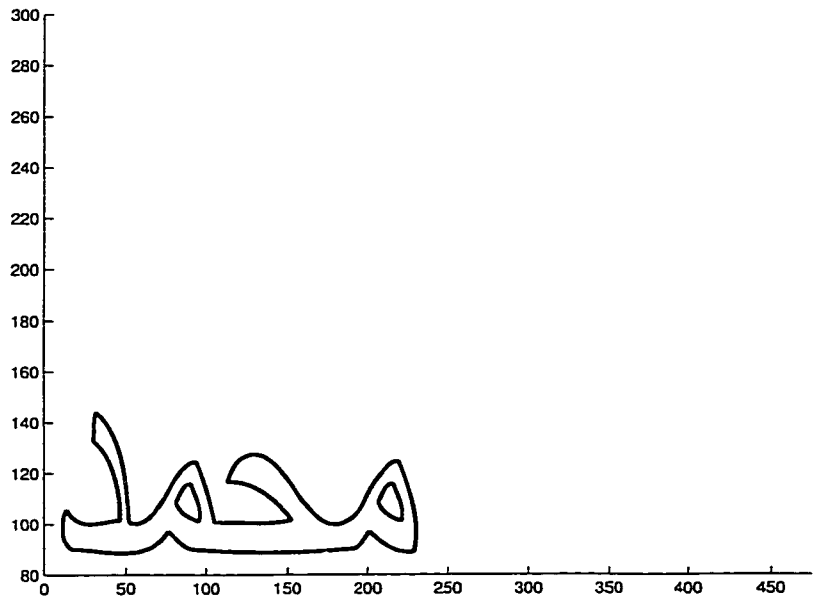


Figure 6.6: Scaling along x-axis and y-axis. $s_x = 1/2, s_y = 1/2$

6.2 Mapping Parametric Surfaces

Mapping of parametric surface can be described in terms of the mapping of a two parameter planar surface in uw parametric space into three-dimensional xyz object space. A surface in object space is represented by the functions that map parametric surface into xyz object space, i.e.

$$\begin{aligned} x &= x(u, w) \\ y &= y(u, w) \\ z &= z(u, w) \end{aligned} \tag{6.3}$$

6.2.1 Two-Dimensional Surface Mapping

Let u and w arrays contain the control points of a segment i.e.

$$u = \begin{bmatrix} P_{x_0} & P_{x_1} & P_{x_2} & P_{x_3} \end{bmatrix}; \quad v = \begin{bmatrix} P_{y_0} & P_{y_1} & P_{y_2} & P_{y_3} \end{bmatrix}$$

We map this parametric space into object space by following functions.

$$\begin{aligned} x &= u - w \\ y &= 2u + w \\ z &= 0 \end{aligned} \tag{6.4}$$

Note that in equation (6.4) z=constant=0, the surface in object space is lying in the z=0 plane.

Figure (6.7) shows the Parametric space of "Alhasab" character. We applied two-dimensional mapping functions of equation (6.4) to all control points of all the segments of each piece of "Alhasab" character and fit Cubic Bezier to mapped significant points. Figures (6.8) to (6.13) show plot of Object space with various *elevation* (El) and *azimuth* (Az) angles [1].

The examples of two-dimensional surface mapping show that holding a single parametric value constant (i.e. z), yields a curve on the surface of Object space. The curve is called *isoparametric.*

6.2.2 Three-Dimensional Surface Mapping

Similarly we can apply three-dimensional mapping to map parametric space into object space. In this case mapping functions are following:

$$\begin{aligned} x &= u \\ y &= w \\ z &= (u - w)^2 \end{aligned} \tag{6.5}$$

Figures (6.14) and (6.15) shows 3-D surface mappings.

[1]The angle orientation with respect to the z=0 plane is called *elevation* and the angle with respect to the x=0 plane is called *azmiuth.*

Our selection of mapping functions is based on following criteria:

- The legibility of character should be preserved.
- A continuous segment must not mapped to discontinuous segment.

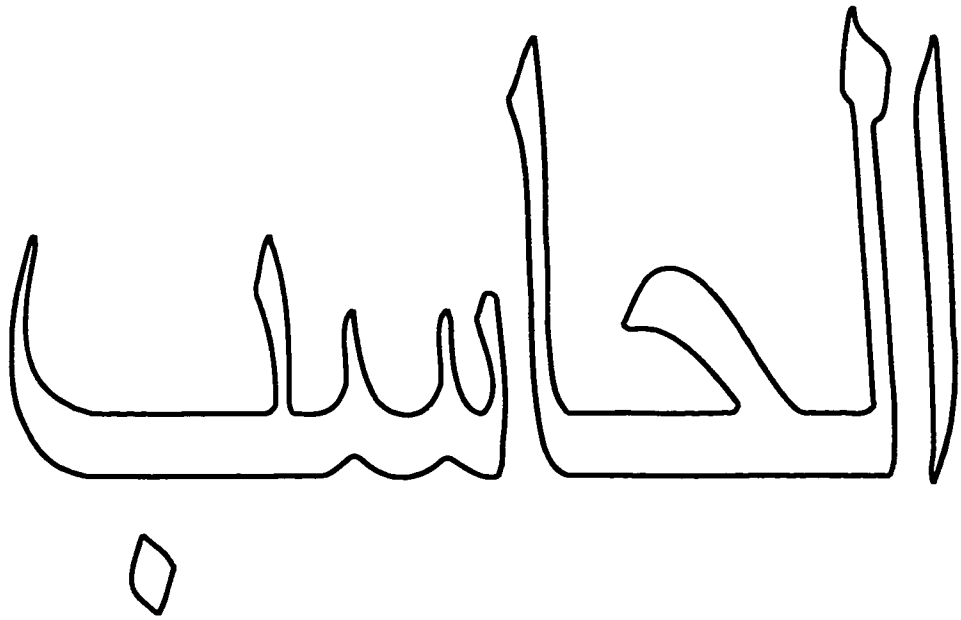


Figure 6.7: Parametric space

Figure 6.8: 2-D Sufrace Mapping. Object space. $Az = 30^o$, $El = 30^o$

Figure 6.9: 2-D Sufrace Mapping. Object space. $Az = 45^o$, $El = 45^o$

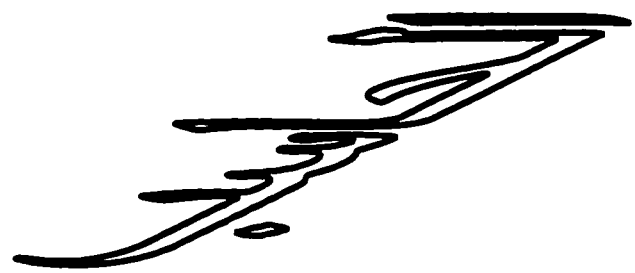


Figure 6.10: 2-D Sufrace Mapping. Object space. $Az = 160^o$, $El = 160^o$

Figure 6.11: 2-D Sufrace Mapping. Object space. $Az = 210^o$, $El = 210^o$

Figure 6.12: 2-D Sufrace Mapping. Object space (Mirrored view). $Az = -120^o$, $El = -60^o$

Figure 6.13: 2-D Sufrace Mapping. Object space. $Az = 330^o$, $El = 330^o$

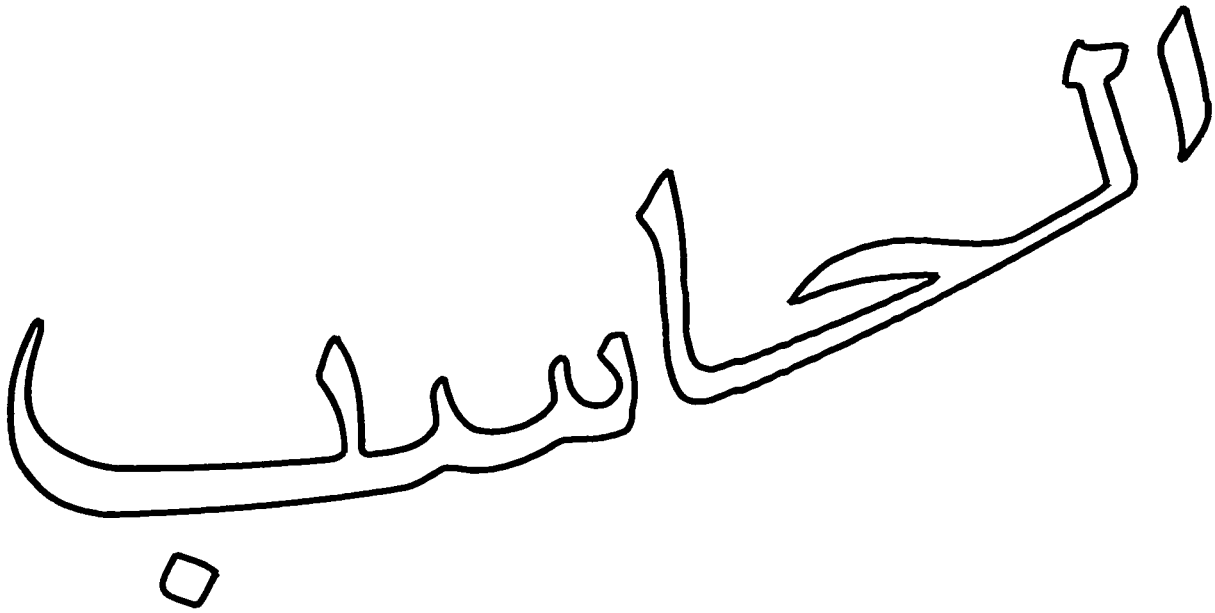


Figure 6.14: 3-D Sufrace Mapping. Object space. $Az = 0^o$, $El = 55^o$

Figure 6.15: 3-D Sufrace Mapping. Object space. $Az = -40^o$, $El = 45^o$

Chapter 7

CONCLUSION AND FUTURE WORK

An efficient method of font design is developed. We tested and presented results in the context of Arabic language but the method is equally applicable for other languages.

The most important aspect of our method is automation of capturing outline of fonts [27] and [28]. We designed the system in a modular way. The output of one module is input to next module. The steps of our system are:

- Getting digitized image
- Extracting Contour (Boundary)
- Detecting Corner Points

- Filtering Noise
- Fitting Cubic Bezier to Significant Points (This is an iterative Process which involves Reparameterization and explring Significant Points)

The objective of our curve fitting process is to approximate the digitize curve with parametric curve in the best way. Our first attempt is to fit the curve with only corner points, if the maximum square distance between digitize curve and parametric curve is greater than desired tolerance limit then we go for reparameterization. If reparameterization also fails to achieve desired tolerance then we ultimately break the segment.

Because of modular design approach any module can be modified without effecting other modules of the system. This makes our system flexible and adaptable for future enhancements.

7.1 Future Work

- **Joining individual alphabets**: For non-roman languages like Arabic and Urdu, words are written by joining different forms of alphabets. Our Font Design System can capture outlines of words or different forms of alphabets. But if someone wants to use this system in a word-processor like application then further work of joining individual is need to be done.

- **Interactive Editing**: Our Font Design System does not provide the interactive editing ability of characters to user. Infact our objective was to eliminate the user intervention in the font design process. But it is possible to extend the work and make it possible to provide interactive editing capability to user.

- **Distributed/Parallel processing**: The nature of our system makes it suitable for distributed/parallel processing. If the character has more than one piece then distributed/prallel processing can be applied at piece level. Corner detection, noise filtering and breaking of piece into segments,fitting curve to segments can be done separately for each piece. It is also possible to apply distributed/parallel processing at segment level. Once a piece is divided into segments, based on corner points. Then further subdivision of each segment is independent of other segments for the same piece. Threads can be used to process each segment (based on corner points).

- **Going for three dimensional**:It will be interesting to investigate and extend the method so that it can work for 3D images.

- **Higher Order Bezier curve**:We used cubic Bezier in our method. It is possible to use Bezeir of degree four or five. Although using higher order Bezier curves will be expensive in terms of computation, but it can be worth to investigate if higher order Bezeir curve can reduce the number of segments required to produce the outline.

- **Other parametric curves**:We find parametric Bezier curve most suitable for capturing outline of fonts. But other parametric curves e.g. B-spline or Hermite models can be used in future. In case of B-spline the selection of knots/control points can be based on genetic algorithm.

Nomenclature

d_{BP}	maximum tolerable square distance between digitized and parametric curve
P_0 and P_3	End-Control Points of Bezier curve
P_1 and P_2	Intermediate-Control Points of Bezier curve
$B_i^3(t)$	cubic Bernstein polynomials $0 \leq i \leq 3$
p_k^+	k^{th} neighbor of point p in clockwise direction
p_k^-	k^{th} neighbor of point p in anti-clockwise direction
d_{min}^2	minimum allowed square distance between p and p_k^+ or between p and p_k^-
d_{max}^2	maximum allowed square distance between p and p_k^+ or between p and p_k^-
cp_i	i^{th} corner point, shown by $\circ$
α_{max}	threshold angle for detecting corner point.
$IndexLimit$	two adjacent corners should be atleast IndexLimit indices away.
$NeighborMax$	maximum number of neighbor points to be checked for testing point p as corner point.

N	number of boundary points in a piece
n	number of points in a segment
S	sum of squared distances from digitized curve to parametric curve.
d_k^2	square distance between k^{th} points of digitized curve and parametric curve
d_{max}^2	maximum square distance between digitized curve and parametric curve
SP	Significant Points (Corner Points and Break Points)
BP	Break Points, shown by $\triangle$
$Q'(t)$	Derivative of parametric curve $Q(t)$

Bibliography

[1] Jean-Pierre Braquelaire and Anne Vialard. A new antialiasing approach for image composition. *The Visual Computer*, 13(5):218–227, 1997.

[2] Dmitry Chetverikov and Zsolt Szabo. A simple and efficient algorithm for detection of high curvature points in planar curves. *Proc. 23rd Workshop of the Australian Pattern Recognition Group*, pages 175–184, 1999.

[3] Won L Chong. Automatic curve fitting using an adaptive local algorithm. *ACM Transactions on Mathematical Software*, 6(1):45–57, March 1980.

[4] E. R. Davies. Locating objects from their point features using an optimal hough-like accumulation technique. *Pattern Recognition Lett*, 13:113–121, 1992.

[5] Victor A. Debelov and Aleksandr M. Matsokin. Implementation of set operations and intersection of bezier curves. *Computer & Graphics*, 24(1):53–65, Februrary 2000.

[6] P. Dierchx. Algorithms for smoothing data with periodic and parametric splines. *Computer Vision, Graphics and Image Processing*, 1982.

[7] L. Dreschler and H. Nagel. Volumetric model and 3d trajectory of a moving car derived from monocular tv frame sequence of a street scene. *Proc. IJCSI*, pages 692–697, 1982.

[8] Roger D. Hersch Ed. *Visual and Technical Aspects of Type.* Cambridge University Press, 1993.

[9] J. El-Sana, F. Evans, A. Kalaiah, A. Varshney, S. Skiena, and E. Azanli. Efficiently computing and updating triangle strips for real-time rendering. *Computer-Aided Design*, 32(13):753–772, November 2000.

[10] Foley, Van Dam, Feiner, and Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley Publishing Company, 1999.

[11] Foley, Van Dam, Feiner, Hughes, and Phillips. *Introduction to Computer Graphics.* Addison Wesley Publishing Company, 1997.

[12] A. E. Fabrisand A. R. Forrest. Antialiasing of curves by discrete pre-filtering. SIGGRAPH 97 Proceedings (Los Angeles, CA, August 1997.

[13] Jean Gallier. A simple method for drawing a rational curves as two bezier segments. *ACM Transactions on Graphics*, 18(4):316–328, October 1999.

[14] M. Grossman. Parametric curve fitting. *The Computer Journal*, 14(2):169–172, 1970.

[15] M. H. Han and D. Jang. The use of maximum curvature points for recognition of partially occluded objects. *Pattern Recognition*, 23:21–23, 1990.

[16] Kozo Ichida, Takeshi Kiyono, and Fujiichi Yoshimoto. Curve fitting by one pass method with a piecewise cubic polynomial. *ACM Transactions on Mathematical Software*, 3(2):164–174, June 1977.

[17] Koichi Itoh and Yoshio Ohno. A curve fitting algorithm for character fonts. *Electronic Publishing*, 6(3):195–198, September 1993.

[18] J.M.Pereira, C.A.Wuthrich, and M.R.Gomes. Full frame merging for sort last polygon rendering computer. The Fourth International Conference in Central Europe on Computer Graphics and Visualization'96 (WSCG'96), Feb 1996.

[19] Shao Lejun and Shou Hao. A new contour fill algorithm for outlined character image generation. *Computer & Graphics*, 19(4):551–556, July/August 1995.

[20] H. C. Liu and M.D. Srinath. Corner detection from chain-code. *Pattern Recognition*, pages 51–68, 1990.

[21] Avi C. Naiman. Jagged edges: When is filtering needed? *ACM Transactions on Graphics*, 17(4):238–258, October 1998.

[22] Bezier P. *Emploi des Machines a Commande Numerique/Numerical Control - Mathematics and Applications.* Wiley, 1972.

[23] Bezier P. *Mathematical and Practical Possibilities of UNISURF in Barnhill, R.E., and R.F. Riesenfeld, eds. Computer Aided Geometric Design.* Academic Press, New York, 1974.

[24] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. *Computer Graphics*, 17(3):229–239, July 1983.

[25] Azhar Quddus. *Curvature Analysis Using Multiresolution Techniques.* PhD thesis, K.F.U.P.M, 1998.

[26] Christian H. Reinsch. Smoothing by spline functions. *Numerical Mathematik*, pages 177–183, 1967.

[27] M. Sarfraz, M. N. Haque, and M. A. Khan. Capturing outlines of 2d images. *PDPTA' 2000 Conference, Las Vegas, USA.*, 2000.

[28] M. Sarfraz and M. A. Khan. Towards automation of capturing outlines of arabic fonts. *Proceeding of WICS 2000 Workshop*, 2000.

[29] Philip J. Schneider. Phoenix:an interactive curve design system based on the automatic fitting of hand sketched curves. Master's thesis, University of Washington, 1988.

[30] T W Sederbert and T Nishita. Curve intersection using bezier clipping. *Computer Aided Design*, 22(9):538–549, November 1990.

[31] Ariel Shamir and Ari Rapport. Compacting oriental fonts by optimizing parametric elements. *The Visual Computer*, 15(6):302–318, 1999.

[32] Miroslav Trajkovic and Mark Hedley. Fast corner detection. *Image and Vision Computing*, 16(2):75–87, February 1998.

[33] Christine Vercken, Christine Potier, and Sylvie Vignes. Spline curve fitting for an interactive design environment. *Theoretical Foundations of Computer Graphics and CAD*, 1987.

[34] Sheng-Chuan Wu, John F. Abel, and Donald P. Greenburg. An interactive computer graphics approach to surface representation. *Communications of ACM*, 20(10):703–712, October 1977.

[35] Fujio Yamaguchi. A new curve fitting method using a crt computer display. *Computer Graphics and Image Processing*, pages 425–437, 1978.

[36] Mark Yang, Chong-Kyo Kim, Kuo-Young Cheng, Chung-Chin, and S.S. Liu. Automatic curve fitting with quadratic b-spline functions and its applications of computer-aided animation. *Computer Vision, Graphics and Image Processing*, pages 346–363, 1986.

[37] Shouqing Zhang, Ling Li, and Hock Soon Seah. Recursive algebraic curve fitting and rendring. *The Visual Computer*, 14(2):69–81, 1998.

[38] Shouqing Zhang, Ling Li, and Hock Soon Seah. Vectorization of digital images using algebraic curves. *Computer & Graphics*, 22(1):91–101, Jan/Feb 1998.

Vita

- Murtaza Ali Khan.
- Born in Karachi, Pakistan on April 16, 1971.
- Received Bachelor of Engineering (B.E) degree in Electrical Engineering from N.E.D University of Engineering and Technology, Karachi, Pakistan in 1995.
- Joined King Fahd University of Petroleum and Minerals in October 1997.
- Currently two Publications.
- Email: khanalimurtaza@yahoo.com