AN AUTOMATIC GENERATION OF G^1 CURVE FITTING OF ARABIC CHARACTERS USING RATIONAL BEZIER CUBIC WITH WEIGHT ADJUSTMENTS

FATIMAH YAHYA, JAMALUDIN MD ALI, AHMAD ABDUL MAJID, ARSMAH IBRAHIM

Fakulti Teknologi Maklumat dan Sains Kuantitatif, Universiti Teknologi Mara, 40000 Shah Alam, Selangor, Malaysia.
fatimahy2003@yahoo.co.uk

Pengajian Sains Matematik, Universiti Sains Malaysia, 11800 Minden, Pulau Pinang, Malaysia.
jamaluma@cs.usm.my

Pengajian Sains Matematik, Universiti Sains Malaysia, 11800 Minden, Pulau Pinang, Malaysia.
majid@cs.usm.my

Fakulti Teknologi Maklumat dan Sains Kuantitatif, Universiti Teknologi Mara, 40000 Shah Alam, Selangor, Malaysia.
arsmah@tmsk.uitm.edu.my

Abstract: An Arabic font is difficult to fit as it is cursive in character, having varying curves and cusps. Here, the Arabic character is represented as an outline font fitted with rational Bezier cubic curves.

As a method in reverse engineering, the Arabic character is created by way of digitizing an image that already exists and then fitting curves automatically to the outline of the digitized image. The outline font representation is done in several phases – contour extraction of font image, corner point's detection and segmentation and lastly contour segment fitting. A rational cubic Bezier with weights is used in the last step. The weights are adjusted automatically to get curves that are as close as we want to the digitized data points.

This technique can be extended to visualizing outlines of any other images automatically.

Keywords: Curve fitting, outline fonts, corner detection, and contour extraction.

1. Introduction

Font representations in the computer include bitmaps, contours (outline), brush strokes and offset curves. Bitmap representation requires every size font to be tuned to excessive detail to a specific printer. Latin uses outline fonts while non Latin with script-like nature have been represented as outlines, brush strokes or offset curves. A big disadvantage of using outline fonts is that, scaling down to smaller sizes reduces legibility. Arabic font present added difficulties as it are cursive in character, having wavy curves and cusps.

Outline representation of fonts are popular as it can be more freely manipulated i.e. resized, translated, rotated and clipped. But there are a loss of legibility and clarity of form when sizes are reduced because of the overall scaling factor. This however can be somewhat overcome by using different fonts for some selected sizes.

There are two ways to create outline fonts. The first is by digitizing some already designed image of fonts and then fitting curves to the outline and second, by designing the curves itself in the computer.

The first of the method is used here. Traditionally it is done in several phases; contour extraction, corner points' detection and lastly contour segment fitting. The contour extraction involved converting the grey level image to bi-level image. Boundary is detected and some curvature analysis is done to detect corner points. Then curves are fitted between the points, usually some form of Bezier curves. His Ming Yang et al [28] used Bezier curves and the accuracy was checked with a computed curve fitness cost. If it is large, control points will be adjusted. [14] iteratively obtained piecewise parametric cubic polynomials close to data points defined by an error measure based on least squares. [23] fits quadratic algebraic curves by recursive curve fitting and fine tuning. Sarfraz [19] algorithm involve recursively segmenting and fitting cubic Bezier curves until a certain least-square tolerance is satisfied by all fitted curves. It is G^0. Sarfraz [20] further improves [19] by reparameterization before further segmenting in the effort to reduce number of curves.

This paper sees the G^1 fitting of the Arabic letter "qaf" (Fig. 1.)by rational cubic curves. The method consists of two main parts- segmentation and curve fitting. The first part sees the digitized outline or boundary been segmented. The second part fits a G^0 and then a G^1 representation to the segments. The weights of the rational cubics are adjusted to get the G^1 curves to be as close as we want to the digitized data points. The whole process is fully automatic.

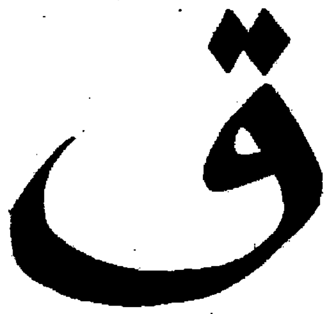


Fig. 1. Digitized image of Qaf

The Algorithm

```
(1) Contour extraction and boundary detection by image analysis.
(2) Segmentation
     (i) Corner detection
    (ii) Breakpoints detection.
(3) Curve fitting.
    (i) $G^0$ representation.
        i.   Fit the first two segments as an optimal cubic Bezier curve.
        ii.  If the optimal curve does not satisfy the tolerance, refit just the first segment.
        iii. Fit the next two unfitted segments.
        iv.  ii. and iii repeated until the whole figure is fitted.
    (ii) $G^1$ representation
        The new segmentation as obtained after (3) (i) iv. is used.
        i.   Segment fitted with joint made $G^1$ with the previous curve.
        ii.  If the curve does not satisfy distance tolerance, frame it in rational Bezier and adjust the
             weights until tolerance is satisfied.
        iii. i. and ii. repeated until whole figure is fitted.
```

Programming is done in Mathematica.

2. Boundary Extraction

Digital images of the Arabic letters are obtained by scanning its designed pictures. The boundary is then extracted. There are many ways to get the boundary, [1], Gonzalez [7] and other good edge detection techniques. Some involve getting it directly from gray level images to minimize errors in detection. However getting the boundary from binary image is simpler and faster. The font image here is considered a "simple" image –i.e. without tremendous detail and is two tone - black characters on white background, justifying the use of binary methods.

However care should be taken to lessen errors in detection. Preprocessing should be done before boundary extraction to smooth out Gaussian noise and to get rid of impulse noise (salt and pepper noise). This can be done using an adaptive median filter without losing much image detail. The resulting digital image is then converted to binary image by thresholding. A closed contour or boundary of the characters is extracted from the binary image. The closed boundary needs to be one pixel thick.

In this paper the digital image of the font obtained from scanning is converted into a binary image. The boundary of the image is then extracted using technique in mathematical morphology, $\beta(A) = A - (A \square B)$ where A is the set of all black pixels, B is the 3x3 structuring element and $\beta(A)$ is the boundary of set A. $\square$ and - represents the operation of erosion and difference respectively.

3. Corner Detection

Corner detection methods falls broadly into 2 categories – ones that act directly on gray levels and those that are based on boundaries. There are two main techniques in gray level methods. Template-based corner detection involves determining similarities between templates of specific angles with all sub-windows of the same size in the image. This technique needed a huge amount of computation. Gradient-based corner detection measures the curvature of an edge that passes through a neighborhood of the gray level image by using the edge strength and gradient of edge direction. This method is not good at localizing corners.

Methods based on boundaries are simpler and faster. Boundaries are represented either by points or chain codes.

Generally chain code methods like [6],[16],[17],[2],[18],[4],[22] and [13] involve computing some measure or estimate of curvature or measure of significance at each point. The corner points are those that are above a certain threshold and are a local maximum

Teh and Chin [24], propose a method that first determines the region of support for each point based on its local properties, and then computes measures of relative significance of each point. Tsai et al [26] propose using eigenvalues of the covariance matrix of a sequence of connected points over a region of support to measure corners. If points form a linear or almost linear line, the small eigenvalue will be close to zero.

Other fields of study have also been applied in techniques to search for the corner points. Tsai [25] uses neural networks, Lin et al[11] and Liu et al[12] use mathematical morphology and Lee and Bien[10] employ fuzzy logic techniques.

It is to be noted that a visually good representation of Arabic fonts, depends much on detecting the right set of corner points.

Here we uses Teh and Chin's [24] method of utilizing region of support with eigenvalues of covariance matrix (see [26]) as a way of measuring corners.

4. Segmentation

The corner points separate sets of data-points into segments. The next step is to find breakpoints by fitting cubic Bezier to the segments formed by the corner points. If any distance between data points and its corresponding fitted curve values is above a specified distance tolerance, the point at which distance is largest is called a breakpoint. These breakpoints will produce more segments and these new segments are again tested for further breaks. The process is repeated until all curves of segments satisfy tolerance limit.

Corner points together with breakpoints are considered as significant points of the outline. They try to describe the twist and turns of the figure. Fig. 2 shows the significant points generated by the above methods. The bigger dark dots denote corner points and the smaller, the breakpoints.

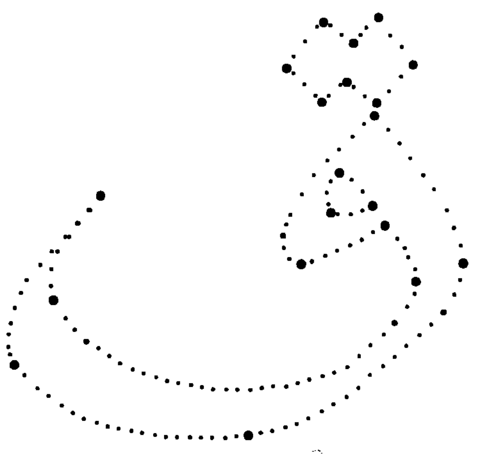


Fig. 2. Digitized outline of Qaf. Dark dots and smaller dark dots denote corner points and breakpoints respectively.

4.1 Parameterization

Chord-length parameterization is used to approximate the value of the parameter t_i for each data point, p_i of each segment. The knot spacing is proportional to the distances of the data points, i.e. $\frac{t_i - t_{i-1}}{t_{i+1} - t_i} = \frac{\| p_i - p_{i-1} \|}{\| p_{i+1} - p_i \|}$. The parameter values t_i for each segment are calculated thus;

$$t_i = \begin{cases} 0 & , \ i = 0 \\ \dfrac{\sum_{j=1}^{i} \| p_{j+1} - p_j \|}{\sum_{j=1}^{m-1} \| p_{j+1} - p_j \|} & , \ 1 \le i \le m-1 \end{cases}.$$

where m = number of data-points in the segment. Note that $t_i \in [0,1]$.

4.2 Determining breakpoints

A parametric curve that best approximate the outline of the segment is fitted to each segment. “Best approximate” means that the sum of squared differences between the fitted curve values and the data-points for each segment is at a minimum. If Q(t) is the fitted curve and p_i the data-points then,

$$S = \sum_{i=1}^{m} (Q(t_i) - p_i)^2 = \sum_{i=1}^{m} (Q_x(t_i) - p_{x_i})^2 + \sum_{i=1}^{m} (Q_y(t_i) - p_{y_i})^2 \ ;$$

is at minimum.

Let $Q(t_i) = B_0(t_i)P_0 + B_1(t_i)P_1 + B_2(t_i)P_2 + B_3(t_i)P_3$ be a cubic Bezier curve. Then it has 4 control points; P_0, P_1, P_2 and P_3. P_0 and P_3 are the first and the last data-points on the segment respectively, which are known. The two intermediate control points are found by minimizing *S*. The x-coordinates of the control points are found by solving the linear system obtained when *S* is differentiated with respect to P_{1x} and P_{2x}. The y-coordinates found by differentiating with respect to y respectively.

A cubic Bezier curve is fitted to the segment with these control points together with P_0 and P_3. ($P_0 = p_1$ and $P_3 = p_m$ where *m* is the number of data points in the segment)

Squared distance between data-points and its corresponding fitted curve values are computed for each segment.

$$d_i^2 = \| p_i - Q(t_i) \|^2 = (p_{ix} - Q_x(t_i))^2 + (p_{iy} - Q_y(t_i))^2$$

Let $d_{max}^2 = \mathrm{Max}(d_1^2, d_2^2, d_3^2, \ldots, d_m^2)$. If d_{max}^2 is greater than a certain tolerance, the segment is divided into two at the point of maximum difference. This point is called the break-point. These new segments will again be fitted with cubic curves. This work is done recursively until tolerance limit is satisfied by all fitted curves. The tolerance limit used will depend on the resolution of output and size of font.

5. G^0 Fitting

The fitting is tried two segments (as defined by the significant points) at a time, so as to get as few curve segments as possible to represent the figure. Fewer segments will give a smoother, better figure of the character. However distance tolerance may not be achieved if the two segments together have more than one curved parts/ inflection and cannot be approximated closely by a cubic. When this happens, fitting is done for the first segment only and then the next two segments still unfitted are considered. This is done repeatedly for the whole figure.

The segment fitting is the same as found in Section 4.2. This piecewise fitting and refitting does not involve complex calculation or much computation. Only two linear systems of two equations need to be solved at each fitting.

6. G^1 Fitting

G^1 curves are fitted between significant points. C^1 is not preferred, as loops may form in order to maintain equal length tangent vectors at joints. In the figure Qaf, there are cusps and very sharp corners which should remain G^0. Their positions are determined at the corner detection stage.

The G^0 curves are forced to become G^1 at joints. But in forcing G^1, the curves do become somewhat distorted and sometimes do not anymore satisfy the distance tolerance, i.e. distance between the Bezier curve at parameter values and the corresponding data points must not be more than the specified distance tolerance of 20 unit squared distance (~ 0.35 mm).

The problem is overcome by framing the curve in rational form with weights w_1, w_2, w_3 and w_4 and then determining these weights that can pull the curve closer towards the digitized data points without changing the control points thus preserving G^1. Fig. 3. shows the curves for the sixth segment of Qaf. The dark curve is the original cubic Bezier, which is G^1 at the joint with the curve before it and do not satisfy the distance tolerance. The light curve is the rational Bezier with weights adjusted to satisfy tolerance. Automatic adjustments and determination of these weights is obtained by optimizing at each iterating reparameterization. The parameterization acts as a design parameter – different parameter values gives different curve. Reparameterization and optimization is done until the desired curve is obtained.

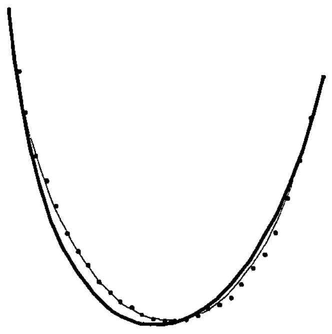


Fig. 3. Sixth segment of Qaf

6.1 *G^1 joints*

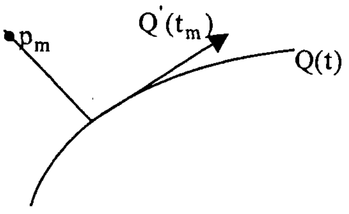
Consider 2 consecutive G^0 Bezier curve segments H(t),Q(t) with control points H_0,H_1,H_2,H_3 ,P_0,P_1,P_2,P_3 where $H_3 = P_0$. These curves need to be made G^1 at P_0. The direction of end tangent of Q (t) is also preserved. *m* and *n* are the length of beginning and end tangent vectors respectively.

$$Q(t) = B_0(t)P_0 + B_1(t)\left(m(H_3 - H_2) + H_3\right) + B_2(t)\left(n(P_2 - P_3) + P_3\right) + B_3(t)P_3$$

$$\text{Let } S^2 = \sum_{i=1}^{n} |Q(t_i) - p_i|^2 = \sum_{i=1}^{n}\left[(Q(t_i)_x - p_{i,x})^2 + (Q(t_i)_y - p_{i,y})^2\right].$$

The value of *m* and *n* that minimize S^2 is found. The new control points are $p_1 = m(H_3 - H_2) + H_3$ and $p_2 = n(P_2 - P_3) + P_3$. The joints are made G^1, piece by piece rather than in a spline as this give simpler calculation. The system above is just a linear system of two equations and is thus very fast.

6.2 *Reparameterization*



Let Q(t) be a cubic Bezier curve and p_m a data point. The curve is assumed to be regular. The shortest distance between them is the perpendicular distance. The parameter corresponding to this shortest distance will give rise to the optimum curve. Let the parameter at which this occurs be t_m. Thus;

$(Q(t_m) - p_m) \cdot Q'(t_m) = 0$, which is quintic in t.

$$\therefore \quad (Q(t_m)_x - p_{m,x})\,Q'(t_m)_x + (Q(t_m)_y - p_{m,y})\,Q'(t_m)_y = 0$$

Let $z = (Q(t)_x - p_{m,x})\,Q'(t)_x + (Q(t)_y - p_{m,y})\,Q'(t)_y$ which again is a quintic.

To simplify calculations, linear interpolation method is used to approximate the solution.

Approximate z to be a linear function of t, i.e. $z = at + b$ for some a and b and for t close to t_m.

Let $z_1 = a t_1 + b$ and $z_2 = a t_2 + b$.

$\therefore\ z - z_1 = a(t - t_1)$ and $z_2 - z_1 = a(t_2 - t_1)$;

$$\frac{z - z_1}{z_2 - z_1} = \frac{t - t_1}{t_2 - t_1}.$$

At $t = t_m$, $z = 0$.

Thus, $\dfrac{t_m - t_1}{t_2 - t_1} = \dfrac{-z_1}{z_2 - z_1}$ and $t_m = \dfrac{z_2 t_1 - z_1 t_2}{z_2 - z_1}$

So by providing t_1 and t_2, the parameter value t_m at which minimum distance occurs can be found iteratively to the degree of precision required. t_m is the new parameter value corresponding to the data point p_m. This method works well if t_1 and t_2 are close to t.

Here the starting t_1 is the original parameter obtained by chord length while t_2 is a step size (0.01) away from t_1. The resulting t from this first iteration becomes t_1 for the next iteration and the t_2 will again be a step size away. The iteration stops when consecutive results for t do not differ in the first three decimal places.

The equation above can of course be solved by Newton Raphson. This linear interpolation method involves simpler calculation requiring at most the evaluation of the first derivative of Q (t) whilst Newton Raphson require up to the second derivative of Q (t).

In Newton-Raphson, $t_2 = t_1 - \dfrac{z(t_1)}{z'(t_1)}$ where t_2 is the updated value of t_1. Computing $z'(t)$ is messy and time consuming to be doing for every point and at each iteration. However Newton-Raphson converges faster than the method above.

6.3 *Iterative procedure to achieve tolerance*

The rational Bezier cubics Q (t);

$$Q(t, w_1, w_2) = \frac{\sum_{i=0}^{3} w_i\, B_i(t)\, P_i}{\sum_{i=0}^{3} w_i\, B_i(t)}$$ where w_0 and w_3 are equal to 1.

Here w_0 and w_3 are made equal to 1 and values of w_1 and w_2 varied. The values of w_1 and w_2 are automatically determined by optimization at each iterating reparameterization.

Let $S^2 = \sum_{i=1}^{n} |Q(t_i, w_1, w_2) - p_i|^2 = \sum_{i=1}^{n}\left[(Q(t_i, w_1 w_2)_x - p_{i,x})^2 + (Q(t_i, w_1, w_2)_y - p_{i,y})^2\right]$ be the squared distance between fitted curve values at parameter values t_i and corresponding digitized data points p_i. Differentiate with respect to w_1 and w_2 and equate to zero.

$$\sum_{i=1}^{n} (Q(t_i, w_1, w_2)_x - p_{i,x})\frac{\partial Q(t_i, w_1, w_2)_x}{\partial w_1} + (Q(t_i, w_1, w_2)_y - p_{i,y})\frac{\partial Q(t_i, w_1, w_2)_y}{\partial w_1} = 0$$

$$\sum_{i=1}^{n}\left(Q(t_i,w_1,w_2)_x - p_{i,x}\right)\frac{\partial Q(t_i,w_1,w_2)_x}{\partial w2} + \left(Q(t_i,w_1,w_2)_y - p_{i,y}\right)\frac{\partial Q(t_i,w_1,w_2)_y}{\partial w2} = 0$$

which is a system of non-linear equations in 2 unknowns ; w_1 and w_2. The system is solved numerically using Newton's Method. This iterative method is started off with w_1 and w_2 taking the value 1. The starting w values for the next reparameterization will use *w* values resulting from the previous iteration. This makes the whole process faster.

7. Results

The G^1 qaf needs nine segments for the main body, three for the inner head and eight for the two dots. All segments in the inner head and dots do not need any iteration to reach tolerance. The table below shows the number of iterations needed by the main body of qaf. Not many iterations are needed to reach tolerance and only four need the iterations in the first place. Fig. 4. shows the G^1 representation of qaf and Fig. 5. shows the curves and the digitized data points it is supposed to approximate.

Table 1. Number of Iterations

Segment	1	2	3	4	5	6	7	8	9
No. of Iterations	0	38	0	4	0	14	0	0	4

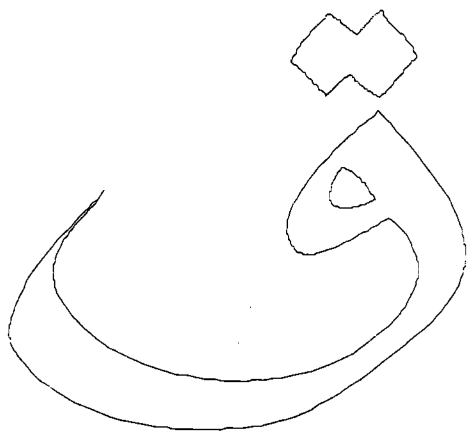


Fig.4. G^1 representation of Qaf

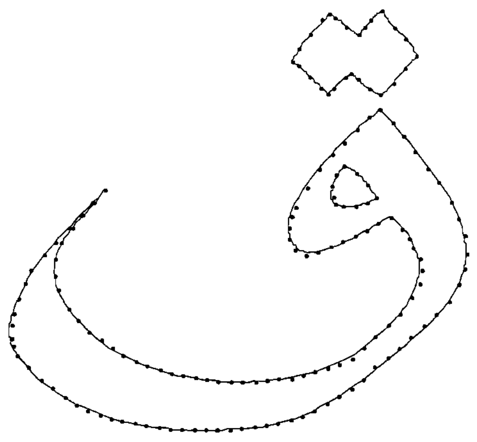


Fig.5. G^1 representation of Qaf with its digitized data points.

8. Conclusion

The G^1 fitting is quite a good approximation and representation of the data points and digitized image respectively, without using massive computation.

References

Avrahami, G. and Pratt, V. (1991). Sub-pixel edge detection in character digitization, *Raster Imaging and Digital Typography* II :54-64.

Beus, H.L. and Tiu, S.H. (1987). An improved corner detection algorithm based on chain coded plane curves, *Pattern Recognition* 20: 291-296.

Burden, R.L. and Faires, J.D. (2001). Numerical Analysis, Seventh Edition. *Brooks/Cole*

Cheng, F. and Hsu, W. (1988). Parallel algorithm for corner finding on digital curves, *Pattern Recognition Letters*, 8: 47-53.

Farin, G. (1990). Curves and surfaces for computer aided geometric design- A practical guide 2nd Edition, *Academic Press*

Freeman, H. and Davis, L.S. (1977). A corner finding algorithm for chain coded curves, *IEEE Trans. Comput.* Vol. C-26: 297-303.

Gonzalez, R.C. and Woods, R.E. (2002). Digital Image Processing 2nd Edition, *Prentice Hall.*

Guru, D.S. and Dinesh, R. (2004). Non parametric adaptive region of support useful for corner detection; a novel approach, *Pattern Recognition*, 37:165-168.

Langridge, D. (1972).On the computation of shape, *Frontiers of Pattern Recognition, S. watanabe, Ed. New York Academic* .:347-365.

Lee, K.J. and Bien, Z. (19996). Grey level corner detector using fuzzy logic, *Pattern Recognition Letters.* 17:939-950.

Lin, R.S., Chu, C.H. and Hsueh, Y.C. (1998). A modified morphological corner detector, *Pattern Recognition Letters*, 19:279-286.

Liu, W.Y., Li, H. and. Zhu, G.X (2001). A fast algorithm for corner detection using the morphologic skeleton, *Pattern Recognition Letters*. 22: 891-900.

Medioni, G. and Yasumoto, Y. (1987). Corner detection and curve representation using cubic B-splines, *Computer Vision, Graphics and Image Processing*, 39: 267-278.

Michael,P. and Maureen S. (1983). Curve fitting with piecewise parametric cubics. *Comput. Graph.* 17 (3) 229-239

Rogers, D.F. and Adams, J.A. (1990). Mathematical elements for computer graphics 2nd edition, *McGraw -Hill.*

Rosenfeld, A.and Johnson, E. (1973). Angle detection on digital curves, *IEEE Trans. Comput.* Vol. C-22: 875-878.

Rosenfeld, A. and Weszka, J.S. (1975). An improved method of angle detection on digital curves, *IEEE Trans. Comput.* Vol.C-24: 940-941.

Rutkowski, W.S. and Rosenfeld, A.(1978) A comparison of corner-detection techniques for chain coded curves, *Technique Report TR-623*,Computer Science Center, University of Maryland.

Safraz, M. and Khan, M.A.(2002). Automatic outline capture of Arabic fonts, *Information Sciences* 140: 269-281.

Safraz, M. and Khan, M.A (2004) An Automatic algorithm for approximating boundary of bitmap characters., *Future Generation Computer Systems* 20: 1327-1336.

Salomon, D. (1999). Computer graphics & geometric modeling. *Springer*

Sankar, P.V. and Sharma, C.V. (1978). A parallel procedure for the detection of dominant points on a digital curve, *Computer Graphics and Image Processing*, 8: 403-412.

Shouqing, Z. , Ling,L, and Hock Soon, S. (1998). Recursive curve fitting and rendering. Visual Comput. 69-82

Teh, C.H. and Chin, R.T. (1989).On the detection of dominant points on digital curves, *IEEE Trans. on pattern analysis and machine intelligence*, 11:859-872.

Tsai, D.M. (1997).Boundary based corner detection using neural networks, *Pattern Recognition*, 30(1):85-97.

Tsai, D.M. and Hou, H.T. and Su, H.J. (1993).Boundary based corner detection using eigenvalues of covariance matrices. *Pattern Recognition Letters*, 20:31-40.

Wolfram, S.(2003) Mathematica Book 5th Edition, *Wolfram Media.*

Yang, H.,Lu, J. and Lee,H. (2001) A Bezier Curve-based Approach to Shape Description for Chinese Calligraphy Characters, *IEEE* :276-280.