GRAPHICAL MODELS AND IMAGE PROCESSING
Vol. 58, No. 3, May, pp. 223–232, 1996
ARTICLE NO. 0019

Curve Fitting with Bézier Cubics

LEJUN SHAO[1] AND HAO ZHOU

School of EEE, Nanyang Technology University, Nanyang Avenue, Singapore 2264, Singapore

Received February 8, 1994; revised January 16, 1996; accepted January 30, 1996

In this paper, a new curve-fitting algorithm is presented. This algorithm can automatically fit a set of data points with piecewise geometrically continuous (G^1) cubic Bézier curves. The algorithm consists of two steps. During the first step, significant points are identified from the given data set and are further classified as either corners or joints. Curve fitting is done in the second step. A weighted least-squares technique is used to find an optimal solution for the construction of piecewise Bézier curves. The resulting Bézier curve segments will be smoothly connected at all joint points. This algorithm has been applied to many digital images with good results.

1. INTRODUCTION

Curve fitting has found many applications in image processing, computer graphics, pattern recognition, and computer-aided geometric design. Our particular interest is in the problem of converting bitmapped images (such as fonts) into its outlined representations, in particular, cubic curve representations. Cubic Bézier curves and B-splines are two of the most frequently used cubic curves [1]. Our study will focus on the use of cubic Bézier curves to fit images' outlines.

Curve fitting may refer to either interpolation or approximation. In this type of applications, we need to find the curves to pass certain data points (endpoints or corner points, for example) and close to others. Therefore, it is an approximation problem. The objective of the curve fitting algorithm is to use a minimum number of cubic curve pieces to approximate the image's outline with minimum distortion.

The ideal case is to use only one piece of cubic Bézier curve to fit the complete outline. This is, however, not possible in most situations. We have to realize the limitations of using cubic Bézier curve to fit image's outlines [3, 8, 9, 11]. Some of the limitations are listed below

1. An image's outline cannot be fitted by a single cubic Bézier curve piece if it contains corners. Corners are points at which the outline's directions take a sharp turn, or the outlines have discontinuous tangent values.

2. An image's outline cannot be fitted by a single cubic Bézier curve if it contains more than two inflection points.

3. An image's outline is difficult to be fitted by a single piece of cubic Bézier curve if it has large curvature changes along certain portion of the curve.

Therefore, in most cases, an image's outline will be fitted by using a piecewise approximation approach. A piecewise approximation to a shape consists of a number of cubic curves pieces connected end-to-end. The location and smoothness of the joints or knots between the pieces is critical to getting a good representation of the image's outline.

A fair amount of research work has been reported in literature [4, 8–11] and there are several ways for identifying the knots to produce piecewise curve approximation. One simple method for defining the knots is to "grow" the curve piece out until the fit error for that piece exceeds a certain threshold [11]. In other words, starting from the previous knot, keep adding data points until the curve piece is as long as possible. Subdivision is another choice [8]. In this method, a single piece of curve is tried to fit the whole set of data points. If the resulting curve gives too big an error, the data points would be divided into two subsets. The breaking point at subdivision step is the point of the greatest fitting error. This point is called knot. This process continues until each subset of data points can be fitted by a curve piece within a given error margin. Both methods are quite sensitive to local noisy data and badly defined tangents.

In theory, to find the best solution, we could try all possible arrangements of knots and take the best fit. However, since there are so many combinations of the arrangements, this solution is clearly useless in practice.

In our study, a new algorithm is developed to find an optimized solution. This algorithm divides the curve fitting process into two steps: *critical point identification* and *curve fitting*.

[1] To whom correspondence should be addressed at DSPT Technology, Inc., 795 Highland Drive, Ann Arbor, MI 48108-2232. E-mail: jshao@dspt.com.

1077-3169/96 $18.00

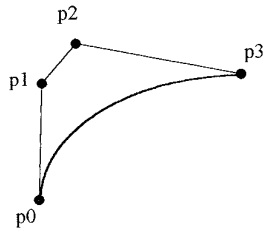

FIG. 1. A cubic Bézier curve segment.

In the critical point identification stage, *an area-deviation algorithm* is developed to calculate the curvature of the digitized curve at each point. During this process, two types of critical points are located: corners and joint points. Between any adjacent pair of critical points, one piece of cubic Bézier curve will be used to fit the subset of data points, we will not force tangent continuity for the curve pieces connected at a corner point. The curve pieces, however, will be smoothly connected at any joint points. The curve pieces between any pair of adjacent corner points are called a curve segment. Each curve segment will be further divided into curve sections by the joint points.

Once the critical points are located, a new curve fitting algorithm called *global optimization fitting* is used in step two to find a best fit of data points between two adjacent corners (curve segments).

In the following, we will first briefly describe the Bézier curves and the important properties which are useful for our fitting algorithm. In Section 3, the corner and joint point finding method will be presented. The recursive least-squares fitting algorithm is discussed in Sections 4 and 5. Experiment results and the conclusion will be given in the last two sections.

2. DEFINITION AND PROPERTIES OF BÉZIER CURVES

A parametric Bézier curve piece of degree n is defined as

$$Q(t) = \sum_{i=0}^{n} \mathbf{V}_i B_i^n(t), \quad 0 \le t \le 1,$$

where the $\mathbf{V}_i$ are the control points and the B_i^n are the Bernstein polynomial.

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

where $\binom{n}{i}$ is the binomial coefficient. For cubic Bézier curves, $n = 3$. See Fig. 1 for an example of a cubic Bézier curve segment.

Cubic Bézier curves have many important properties. Some of them are useful in implementing our algorithm and are given as follows:

1. $\mathbf{V}_0$ and $\mathbf{V}_3$ are the two endpoints of the curve segment. This can be easily proved if we set $n = 3$, $t = 0$, and $t = 1$, respectively in the above formula.

2. A straight line segment is a special case of cubic Bézier curve. It can also be represented by a Bézier curve. This can be done by selecting any two points on the straight line segment $\overline{\mathbf{V}_0\mathbf{V}_3}$ as control points $\mathbf{V}_1$ and $\mathbf{V}_2$.

3. $\mathbf{V}_1$ and $\mathbf{V}_2$ are the two inner control points and are located on the tangent vectors of the curve at $\mathbf{V}_0$ and $\mathbf{V}_3$, respectively. That is, if k_0 and k_3 are scalars, we have

$$\mathbf{V}_1 = \mathbf{V}_0 + \Delta\mathbf{V}_0 * k_0$$
$$\mathbf{V}_2 = \mathbf{V}_3 + \Delta\mathbf{V}_3 * k_3.$$

If two Bézier curve segments are smoothly connected at their joint, the two tangent vectors at the joint must have the same direction. We call this geometric continuity (G') (see Fig. 2).

3. CRITICAL POINT IDENTIFICATION

The critical points identification phase plays an important role in the curve fitting algorithm in this study. It provides knots to assist the fitting process. Two types of knots are identified during this process: corners and joint points.

Many corner detection algorithms have been reported in the literature. In Liu and Srimath's [7] paper, six corner detection algorithms based on chain-code were discussed. The researchers used a set of test images to judge the goodness of those algorithms and found that the Beus–Tiu's algorithm [5], which is an improvement of Freeman–Davis' algorithm [6], gave better results for all samples. Shao [12] implemented Beus–Tiu's algorithm to detect corners in many images and found that Beus–Tiu's algorithm was not suitable for detecting corners of rough, uneven boundaries. An improved algorithm for corner detection based on Beus-Tiu's algorithm was also presented by Shao.

There are not many papers in the literature that deal with joint point detection. Some papers dealing with dominant

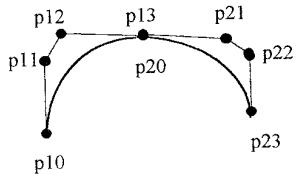

FIG. 2. G^1 continuous Bézier curves.

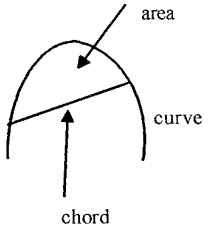

FIG. 3. Chord, area, and curve.

point detection can be found in [2, 3, 14]. Pavlidis [8] described a method to classify a vertices as hard, soft, or break according to a certain angular criteria.

The main difference between corners and joint points is that the corners are those points having tangent discontinuity, whereas the joint points are those points having big curvature changes. The two types of points are also called critical points. In our study, we have developed a new algorithm which can detect both corners and joints at the same time. The algorithm is based on the computation of curvature values.

Suppose a curve is expressed as $y = f(x)$. The curvature k at a point (x, y) on the curve is defined as

$$k = \frac{y''}{\sqrt[3]{(1 + (y')^2)^2}}. \tag{1}$$

The critical points are usually related to the points having maximum curvature values or points having a sharp change in their curvature values. For digitized curves, however, Eq. (1) cannot be used. In our algorithm, the area values between a series of fixed-length chords and curve pieces along the digitized curve are calculated and used to represent the average curvature values for the corresponding curve pieces (Fig. 3). They are also indicators of curve direction change. The curve pieces which may contain potential critical points are first identified based on their area values; the exact location of the critical points within each curve piece will then be determined.

A more detailed description of the two-step algorithm will be given in the following two sections and a discussion about how to select parameters is given in the last section.

3.1. *Critical Point Detection*

In this step, approximate locations of the possible critical points are identified. A fixed-length chord is moving along the digitized curve step by step, and the area between the chord and the associated curve piece is measured. This area, called the chord-curve area, is used in the algorithm to help to identify the approximate locations of the possible critical points.

Calculating Chord–Curve Areas. Suppose that the digitized curve contains n points, represented as $P_0, P_1, \ldots, P_{n-1}$, and the length of the fixed-length chords is L. Starting from the first point P_0, compute the length of chord P_0P_1. If P_0P_1 is smaller than L, compute the length of chord P_0P_2. If length of P_0P_2 is still smaller than L, move to the next point P_3.

This process will stop at a point Pm_0 where the length of chord P_0P_{m0} is equal or greater than L. Then P_{m0} is the terminal point of the first fixed-length chord. Calculate the area between the chord P_0P_{m0} and the associated curve piece. If the coordinates for point P_k is $P_k(x_k, y_k)$, the formula to compute the area is

$$S_{0i} = \begin{vmatrix} x_0 & x_1 & x_2 & \cdots & x_i & x_0 \\ y_0 & y_1 & y_2 & \cdots & y_i & y_0 \end{vmatrix}. \tag{2}$$

It can be expanded to

$$S_{0i} = \sum_{p=0}^{i} (x_p y_{p+1} - x_{p+1} y_p) + x_i y_0 - x_0 y_i. \tag{3}$$

Next, move the start point from P_0 to P_1 and look for the end point P_{m1}. It may be in point P_{m0+1}, in point P_{m0+2}, etc., so that the length of P_1P_{m1} is equal to or greater than L. The chord-curve area for P_1P_{m1} is calculated. Then, move the start point from P_1 to P_2, look for the third fixed length-chord P_1P_{m2}, and calculate the associated chord-curve area. This process is repeated until every point on the curve has each become the start point for a chord. All the chord-curve areas for the sequential chords are then calculated.

Starting from the second point, an incremental method will be used to compute the area. In general, suppose that the chord-curve area S_{ij} from point P_i to point P_j has been

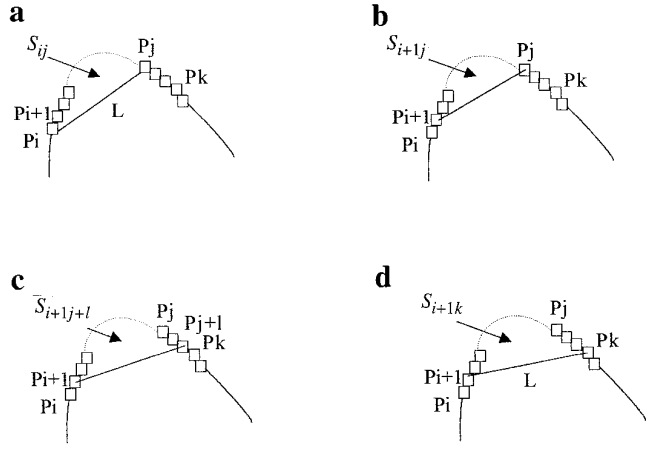

FIG. 4. Calculation of the chord-curve area.

calculated (Fig. 4a), we want to compute the chord-curve area $S_{i+1,k}$, which is from point P_{i+1} to point P_k (Fig. 4d).

First, the start point is moved one step forward from P_i to P_{i+1}, and the algorithm looks at the area change ∇S_{ij}^{i+1} between S_{ij} and area $S_{i+1,j}$ (Fig. 4b), which is a triangular formed by three vertices of P_i, and P_{i+1} and P_j. This area is equal to

$$\nabla S_{ij}^{i+1} = \begin{vmatrix} x_i & x_{i+1} & x_j & x_i \\ y_i & y_{i+1} & y_j & y_i \end{vmatrix}. \tag{4}$$

Next, the end point is moved l steps forward from P_j to P_{j+1}, and we define the area increment from $S_{i+1,j}$ to $S_{i+1,j+1}$ as $\nabla S_{i+1,j}^{j+l}$ (Fig. 4c). Then $\nabla S_{i+1,j}^{j+l}$ can be expressed as

$$\nabla S_{i+1,j}^{j+l} = \begin{vmatrix} x_j & x_{j+1} & \cdots & x_{j+l} & x_{i+1} & x_j \\ y_j & y_{j+1} & \cdots & x_{j+l} & y_{i+1} & y_j \end{vmatrix}. \tag{5}$$

Let $l = k - j$; we get

$$\nabla S_{i+1,j}^{k} = \begin{vmatrix} x_j & x_{j+1} & \cdots & x_k & x_{i+1} & x_j \\ y_j & y_{j+1} & \cdots & x_k & y_{i+1} & y_j \end{vmatrix}. \tag{6}$$

Combine the results of Eqs. (4) and (6), we have

$$\begin{aligned} S_{i+1,k} = S_{ij} &- \begin{vmatrix} x_i & x_{i+1} & x_j & x_i \\ y_i & y_{i+1} & y_j & y_i \end{vmatrix} \\ &+ \begin{vmatrix} x_j & x_{j+1} & \cdots & x_k & x_{i+1} & x_j \\ y_j & y_{j+1} & \cdots & x_k & y_{i+1} & y_j \end{vmatrix}. \end{aligned} \tag{7}$$

The formula in Eq. (7) uses the result from earlier calculations and only area change needs to be computed. This makes the algorithm very efficient.

Identifying the Approximate Position of Corners and Joint Points. Having computed the areas between the fixed-length chords and asociated curve fragments, we will have n chord-curve area values. They correspond to chords P_0P_{m0}, P_1P_{m1}, P_2P_{m2}, ..., P_nP_{mn}. These values will be used to identify potential critical points.

To detect corners, we will scan the area values and look for a local maximum value. If the area values reached a local maximum at P_kP_{mk} and that area value is greater than a predefined threshold value V_c that curve fragment will be selected as a candidate fragment.

After corners are detected, we need to locate the rough position of joint points. We took a different approach to detect joint points. This approach is based on area deviations. The area deviation d_i of chord P_iP_{mi} is the difference between the area values of $P_{i-1}P_{mi-1}$ and $P_{i+1}P_{mi+1}$. The absolute value of d_i measures the curvature changes in the neighborhood of the curve fragments C_{i-1}, C_i, and C_i+1 which are associated with the chords L_{i-1}, L_i, and L_{i+1}. If d_i is bigger than a predefined threshold value V_j, it implies that the curvature change in the neighborhood of the curve fragment C_i is big, we can say that there is a joint point in the curve fragment C_i. In general, for a typical joint point, there may be several curve fragments in the neighborhood of the joint point such that the associated area deviations are all bigger than the threshold value V_j. The curve fragment having the biggest area deviation will be chosen to represent the joint point. The process to locate the rough positions of the joint points can be summarized as follows: (1) calculating the area deviations; (2) finding out all the local maximums of the area deviations; (3) detecting those local maximums which are bigger than the threshold value V_j and picking up the associated curve fragments; (4) counting the number of joint points in the curve.

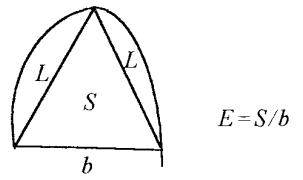

FIG. 5. Compute the direction change at a point.

3.2. *Critical Point Localization*

After the detection step, all the candidate corners and joint points are obtained, and the rough locations of these points are also found. The exact positions of these critical points will be determined in this step. Usually, the chord length value L is quite small. We can simply select the middle point of each selected curve fragment as the exact position of the critical point without much error.

However, to get an accurate location of the critical point within each candidate curve fragment, we can choose the point in the fragment whose direction change is the biggest as the critical point.

For each point on the curve fragments, we calculate the direction changes of two chords starting from the point, one going forward, the other going backward (Fig. 5).

The lengths of the two chords can be chosen the same as L. Let S be the area of the triangle formed by the two chords and an additional line segment joining the other two endpoints of the chords. Assuming the length of the additional line segment is b. Then, the direction change E of the two chords can be expressed as $E = S/b$. Larger E values correspond to smaller angles A formed by the two chords.

For each candidate curve fragment, we look for the point

having the maximum E value and assign it as the corner or joint point. In other words, for each candidate curve fragment detected in the first step, the point having the biggest change in curve direction is identified as the corner or joint point.

3.3. *Parameter Selection*

For the critical points detection algorithm stated in last section to work well, it is important to choose the proper threshold values and the proper length of the fixed-length chord.

The chord length L is related to the size and resolution of the image and the image's noise level.The following formula is used to calculate the chord length:

$$L = \alpha \cdot p \cdot A \cdot \sqrt{ab}.$$

Here p indicates the degree of noise level, A is the image's resolution, a and b are the width and length of the image; and α is a constant.

From the formula, we can see that L is proportional to the noise level, the resolution and the image's size. The threshold value to detect corners V_c and the threshold value to detect joint points V_j are functions of the chord length and the noise level of the image.

In practice, an additional step—*local curve smoothing*—is applied to the data set to filter out jag noises before the critical points detection step. We have identified seven noise patterns and developed a very efficient algorithm for local noise removal. For a more detailed discussion about the smoothing algorithm, please refer to [15].

4. RECURSIVE LEAST-SQUARES CURVE FITTING

Having identified corner and joint points, we are ready to fit curves. Since we do not force tangent continuity for the curves connecting at a corner point, our curve-fitting algorithm needs only to consider fitting piecewise Bézier curves for data points between two adjacent corner points.

The problem now can be stated as follows:

> Given *two* corner points, n-2 joint points and a set of data points, n-1 pieces of cubic Bézier curves need to be found to fit these data points. The corner and joint points should be the two endpoints of corresponding Bézier curve pieces. The Bézier curve pieces should be smoothly connected at all joint points and the solution should be an optimal one.

Suppose, $\mathbf{P}_1$ and $\mathbf{P}_n$ are the two corner points, $\mathbf{P}_i$ ($i \in [2, n-1]$) are the joint points, and Q_i ($i \in [1, n-1]$) are the to-be-derived Bézier curve pieces.

Let $\mathbf{P}_{i0}$ and $\mathbf{P}_{i1}$ ($i \in [1, n-1]$) be the two inner control points of Bézier curve piece Q_i and $\hat{\mathbf{t}}_i$ ($i \in [1, n]$) be the unit tangent vector at endpoint $\mathbf{P}_i$ of the piecewise Bézier curves. We want to find the n-1 pieces of cubic Bézier curves Q_i such that:

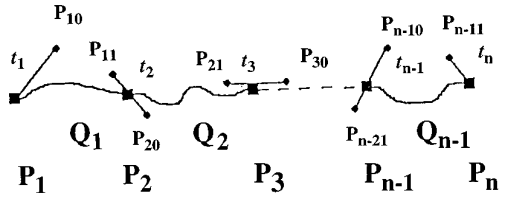

FIG. 6. Use a global optimization method to fit $n-1$ piecewise cubic Bézier curve.

1. the sum of square distance errors from each data point to the curve is minimal;
2. tangents at $\mathbf{P}_i (i \in [2, n-1])$ are continuous.

That is, we need to find Q_i's to minimize S in the expression

$$S = \sum_{i=1}^{n-1} \sum_{j=1}^{m_i} [Q_i(t_{ij}) - C_i(j)]^2, \tag{8}$$

where, m_i represents the number of points to be fitted by ith piece of curve, $C_i(j)$ is the jth data point to be fitted in ith piece of the Bézier curve section.

The ith piece of the cubic Bézier curve is expressed as

$$\begin{aligned} Q_i(t_{ij}) = {} & \mathbf{P}_i B_{13}(t_{ij}) + \mathbf{P}_{i0} B_{23}(t_{ij}) \\ & + \mathbf{P}_{i1} B_{33}(t_{ij}) + \mathbf{P}_{i+1} B_{43}(t_{ij}), \end{aligned} \tag{9}$$

where $B_{13} = (1 - t_{ij})^3$, $B_{23} = 3(1 - t_{ij})^2 t_{ij}$, $B_{33} = 3(1 - t_{ij}) t_{ij}^2$, and $B_{43} = t_{ij}^3$, $t_{ij} \in [0, 1]$.
The inner control points of curve $Q_i(t_{ij})$ are defined by

$$\begin{cases} \mathbf{P}_{i0} = \mathbf{P}_i + \alpha_{i0} \hat{\mathbf{t}}_i \\ \mathbf{P}_{i1} = \mathbf{P}_{i+1} - \alpha_{i1} \hat{\mathbf{t}}_{i+1} \end{cases}$$

where α_{i0} and α_{i1} are scalars. They are used to specify the location of control points $\mathbf{P}_{i0}$, $\mathbf{P}_{i1}$ on the tangent vectors defined by $\hat{\mathbf{t}}_i$ and $\hat{\mathbf{t}}_{i+1}$, respectively. Refer to Fig. 6 for an illustration.

Expanding $Q_i(t_{ij}) - C_i(j)$ in Eq. (8) and defining

$$\begin{aligned} A_i(t_{ij}) = {} & \mathbf{P}_i B_{13}(t_{ij}) + \mathbf{P}_{i0} B_{23}(t_{ij}) \\ & + \mathbf{P}_{i1} B_{33}(t_{ij}) + \mathbf{P}_{i+1} B_{43}(t_{ij}) - C_i(j), \end{aligned} \tag{10}$$

Eq. (8) will become

$$\begin{aligned} S = \sum_{i=1}^{n-1} \sum_{j=1}^{m_i} [A_i(t_{ij}) + \alpha_{i0} \hat{\mathbf{t}} B_{23}(t_{ij}) - \alpha_{i1} \hat{\mathbf{t}}_{i+1} B_{33}(t_{ij})]^2, \\ i \in [1, n-1]. \end{aligned} \tag{11}$$

Our problem is to solve $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$, and $\hat{\mathbf{t}}_i (i \in$

$[1, n]$) to minimize S. This is a non-linear problem, which cannot be solved by traditional least-squares techniques.

To deal with this problem, we break the problem into two subproblems so that each of them will become a linear one. To do that, we give each tangent vector $\hat{\mathbf{t}}_i (i \in [1, n])$ an initial value and try to solve the $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$. This is a linear problem and can be solved by using the traditional least-squares technique.

Once we obtained an initial $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$, we can use the result to solve for $\hat{\mathbf{t}}_i (i \in [1, n])$, again by using a traditional least-squares technique.

By recursively applying the least-squares fit, optimal tangent vectors $\hat{\mathbf{t}}_i (i \in [1, n])$ at each corner and joint point and the optimal scalar values of $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$ can all be obtained.

The initial $\hat{\mathbf{t}}_i (i \in [1, n])$ can be obtained in many ways, such as the least square fitting of tangent lines at the corner and joint points.

In the following, we illustrate how $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$, and $\hat{\mathbf{t}}_i (i \in [1, n])$ can be solved individually.

First, assume that the tangent vectors $\hat{\mathbf{t}}_i (i \in [1, n])$ are known. We want to find $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$. This is a linear problem and the traditional least squares technique can be used:

$$\frac{\partial S}{\partial \alpha_{i0}} = 0, \quad i \in [1, n-1], \tag{12}$$

$$\frac{\partial S}{\partial \alpha_{i1}} = 0, \quad i \in [1, n-1]. \tag{13}$$

Applying Eq. (12) to Eq. (11), we have

$$2 \sum_{j=1}^{m_i} \hat{\mathbf{t}}_i B_{23}(t_{ij}) [A_i(t_{ij}) + \alpha_{i0} \hat{\mathbf{t}}_i B_{23}(t_{ij}) - \alpha_{i1} \hat{\mathbf{t}}_{i+1} B_{33}(t_{ij})] = 0, \quad i \in [1, n-1].$$

Rearranging the terms, we get

$$\begin{aligned} \alpha_{i0} \hat{\mathbf{t}}_i^2 \sum_{j=1}^{m_i} B_{23}^2(t_{ij}) - \alpha_{i1} \hat{\mathbf{t}}_i \hat{\mathbf{t}}_{i+1} \sum_{j=1}^{m_i} B_{23}(t_{ij}) B_{33}(t_{ij}) \\ = -\hat{\mathbf{t}}_i \sum_{j=1}^{m_i} B_{23}(t_{ij}) A_i(t_{ij}), \quad i \in [1, n-1]. \end{aligned} \tag{14}$$

Similarly, applying Eq. (13) to Eq. (11), we have

$$\begin{aligned} \alpha_{i0} \hat{\mathbf{t}}_i \hat{\mathbf{t}}_{i+1} \sum_{j=1}^{m_i} B_{23}(t_{ij}) B_{33}(t_{ij}) - \alpha_{i1} \hat{\mathbf{t}}_{i+1}^2 \sum_{j=1}^{m_i} B_{33}^2(t_{ij}) \\ = -\hat{\mathbf{t}}_{i+1} \sum_{j=1}^{m_i} B_{33}(t_{ij}) A_i(t_{ij}), \quad i \in [1, n-1]. \end{aligned} \tag{15}$$

The previous two equations can be simplified as

$$\begin{cases} d_{i1}\alpha_{i0} + d_{i2}\alpha_{i1} = x_{i1} \\ d_{i3}\alpha_{i0} + d_{i4}\alpha_{i1} = x_{i2} \end{cases}, \quad i \in [1, n-1].$$

That is, we need only to solve a 2×2 system of linear equations

$$\begin{pmatrix} d_{i1} d_{i2} \\ d_{i3} d_{i4} \end{pmatrix} \begin{pmatrix} \alpha_{i0} \\ \alpha_{i1} \end{pmatrix} = \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix}$$

for each $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1])$.

If we let

$$D \begin{pmatrix} d_{i1} d_{i2} \\ d_{i3} d_{i4} \end{pmatrix} = (D_{i1} D_{i2})$$

and

$$X_i = \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix},$$

the solution will be

$$\alpha_{i0} = \frac{\det(X_i D_{i2})}{\det(D_{i1} D_{i2})} \tag{16}$$

and

$$\alpha_{i1} = \frac{\det(X_i D_{i1})}{\det(D_{i1} D_{i2})}. \tag{17}$$

Now, assume that $\alpha_{i0}, \alpha_{i1} (i \in [1, n-1)]$ are given; we want to find tangent vectors $\hat{\mathbf{t}}_i (i \in [1, n])$. We take partial differentiation of Eq. (11) with respect to $\hat{\mathbf{t}}_i$ and set it to zero:

$$\frac{\partial S}{\partial \hat{\mathbf{t}}_i} = 0, \quad i \in [1, n]. \tag{18}$$

When $i = 1$, we have

$$\frac{\partial S}{\partial \hat{\mathbf{t}}_i} = 0. \tag{19}$$

Applying Eq. (19) to Eq. (11), we have

$$\alpha_{10} \sum_{j=1}^{m_i} B_{23}(t_{1j}) [A_1(t_{1j}) + \alpha_{10} \hat{\mathbf{t}}_1 B_{23}(t_{1j}) - \alpha_{11} \hat{\mathbf{t}}_2 B_{33}(t_{1j})] = 0$$

Rearranging the terms, we get

$$\hat{\mathbf{t}}_1\alpha_{10}^2\sum_{j=1}^{m_i}B_{23}^2(t_{ij})-\hat{\mathbf{t}}_2\alpha_{10}\alpha_{11}\sum_{j=0}^{m_i}B_{23}(t_{1j})\,B_{33}(t_{1j})$$
$$=-\alpha_{10}\sum_{j=0}^{m_i}B_{23}(t_{1j})A_1(t_{1j}). \tag{20}$$

When $i = n$, we have

$$\frac{\partial S}{\partial\hat{\mathbf{t}}_n}=0 \tag{21}$$

If we go through a similar procedure, we have

$$\hat{\mathbf{t}}_{n+1}\alpha_{n-10}\alpha_{n-11}\sum_{j=1}^{m_{n-1}}B_{23}(t_{n-1j})B_{33}(t_{n-1j})$$
$$-\hat{\mathbf{t}}_n\alpha_{n-11}^2\sum_{j=1}^{m_{n-1}}B_{33}^2(t_{n-1j}) \tag{22}$$
$$=-\alpha_{n-11}\sum_{j=1}^{m_{n-1}}B_{33}(t_{n-1j})A_{n-1}(t_{n-1j}).$$

For $i \in [2, n-1]$, equations $\partial S/\partial\hat{\mathbf{t}}_n = 0$ will give us

$$\alpha_{10}\sum_{j=1}^{m_i}B_{23}(t_{ij})[A_i(t_{ij})+\alpha_{i0}\hat{\mathbf{t}}_iB_{23}(t_{ij})-\alpha_{i1}\hat{\mathbf{t}}_{i+1}B_{33}(t_{ij})]$$
$$\alpha_{i-11}\sum_{j=1}^{m_{i-1}}B_{33}(t_{i-1j}) \tag{23}$$
$$[A_{i-1}(t_{i-1j})+\alpha_{i-10}\hat{\mathbf{t}}_{i-1}B_{23}(t_{i-1j})-\alpha_{i-11}\hat{\mathbf{t}}_iB_{33}(t_{i-1j})]=0.$$

Rearranging, we get

$$-\hat{\mathbf{t}}_{i-1}\alpha_{i-10}\alpha_{i-11}\sum_{j=1}^{m_{i-1}}B_{23}(t_{i-1j})B_{33}(t_{i-1j})$$
$$+\hat{\mathbf{t}}_i\left\{\alpha_{i0}^2\sum_{j=1}^{m_i}B_{23}^2(t_{ij})\alpha_{i-11}^2\sum_{j=1}^{m_{i-1}}B_{23}^2(t_{i-1j})\right\}$$
$$-\hat{\mathbf{t}}_{i+1}\alpha_{i0}\alpha_{i1}\sum_{j=0}^{m_i}B_{23}(t_{ij})B_{33}(t_{ij}) \tag{24}$$
$$=-\alpha_{i0}\sum_{j=1}^{m_i}B_{23}(t_{ij})A_i(t_{ij})+\alpha_{i-11}\sum_{j=0}^{m_{i-1}}B_{33}(t_{i-1j})$$
$$A_{i-1}(t_{i-1j}),\quad i\in[2,n-1]$$

For convenience, we can represent Eq. (20) as

$$e_{11}\hat{\mathbf{t}}_1+e_{12}\hat{\mathbf{t}}_2=y_1, \tag{25}$$

represent Eq. (22) as

$$e_{nn-1}\hat{\mathbf{t}}_{n-1}+e_{nn}\hat{\mathbf{t}}_n=y_n, \tag{26}$$

and represent Eq. (24) as

$$e_{ii-1}\hat{\mathbf{t}}_{i-1}+e_{ii}\hat{\mathbf{t}}_i+e_{ii+1}\hat{\mathbf{t}}_{i+1}=y_i,\quad i\in[2,n-1]. \tag{27}$$

Then, we need only solve the following system of linear equations:

$$\begin{pmatrix}
e_{11} & e_{12} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0\\
e_{21} & e_{22} & e_{23} & 0 & 0 & \cdots & 0 & 0 & 0 & 0\\
0 & e_{32} & e_{33} & e_{34} & 0 & \cdots & 0 & 0 & 0 & 0\\
 & & \cdots & & & \cdots & & & \cdots & \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & e_{n-1n-2} & e_{n-1n-1} & e_{n-1n}\\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & e_{nn-1} & e_{nn}
\end{pmatrix}$$
$$\begin{pmatrix}\hat{\mathbf{t}}_1\\ \hat{\mathbf{t}}_2\\ \vdots\\ \hat{\mathbf{t}}_n\end{pmatrix}=\begin{pmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{pmatrix}. \tag{28}$$

The coefficient matrix can be transferred into a triangle matrix and the system of equations is therefore easy to solve.

During the implementation of our recursive algorithm, we have to solve the problem of associating a proper t_{ij} value with each data point. Plass and Stone [9] described a chord-length parameterization method. This method is also used in our algorithm to get the initial values for t_{ij}. This approach, however, is not accurate. Once the initial Bézier curve parameters are obtained, a different approach was used to estimate t_{ij} in order to improve the accuracy of curve fitting. That is, in subsequent iterations, we compute the distance between any data point and the approximated curve and use the computed result as the t_{ij} values. The problem can be stated as follows:

> Given a data point $C_i(j)$, find a point $Q_i(t_{ij})$ on the associated Bézier curve piece to make the distance between the two points minimal.

The nearest points of $C_i(j)$ on Bézier curve piece Q_i can be computed by the following equations:

$$[Q_i(t_{ij})-C_i(j)]\cdot Q_i'(t_{ij})=0. \tag{29}$$

We can use Newton-Raphson iteration to solve for t_{ij}. The general Newton–Raphson iteration formula to solve equation $f(n) = 0$ is

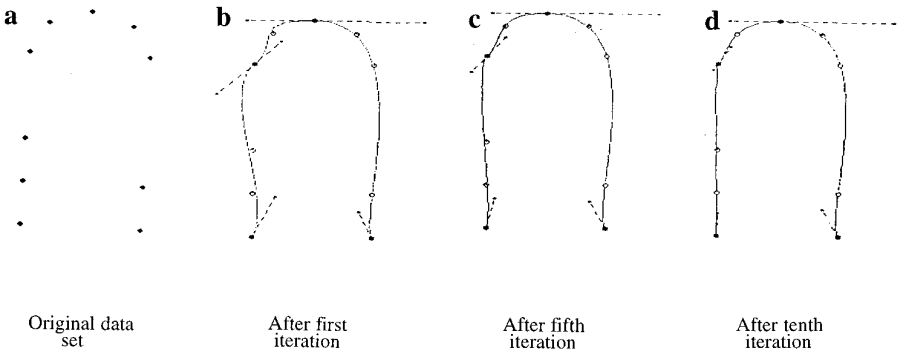

FIG. 7. Curve-fitting process.

$$t \leftarrow t - \frac{f(t)}{f'(t)}. \tag{30}$$

To use the formula in our application, the will be replaced by

$$\frac{[Q_i(t_{ij}) - C_i(j)] \cdot Q_i'(t_{ij})}{\{[Q_i(t_{ij}) - C_i(j)] \cdot Q_i'(t_{ij})\}'} \tag{31}$$

Figure 7 shows the curve-fitting process by using our algorithm. Figure 7a shows the original data set to be fitted. Figures 7b–7d are the result of iteration process. The converge speed is quite high. Our algorithm will exit either the maximum fitting error has been reduced to be less than the maximum allowable error or the total number of iterations has exceeded the maximum allowable iteration steps. In our experiment, *MaxStep* was set between 20 and 40 because the fitting result showed no significant improvement after 40 iterations. A summary of the algorithm is given in Table 1.

5. WEIGHTED LEAST-SQUARES CURVE FITTING

To further improve the tightness of the fitting, we applied different weights on different data points based on their curvature values. That is, we put less emphasis on long flat portion of the curve segment and more weight on short but sharp-turn portion of the curve segment such as points near the corner points. That is, we will add a weigh factor w_{ij} to each data point $C_i(j)$. As a result, Eq. (4) will become

$$S = \sum_{i=1}^{n-1} \sum_{j=1}^{m_i} [Q_i(t_{ij}) - w_{ij} C_i(j)]^2. \tag{32}$$

The equations to compute α_{i0}, $\alpha_{i1} (i \in [1, n-1])$ will be changed to

$$\alpha_{i0} \hat{\mathbf{t}}_i^2 \sum_{j=1}^{m_i} W_{ij} B_{23}^2(t_{ij}) - \alpha_{i1} \hat{\mathbf{t}}_i \hat{\mathbf{t}}_{i+1} \sum_{j=1}^{m_i} W_{ij} B_{23}(t_{ij}) B_{33}(t_{ij}) = -\hat{\mathbf{t}}_i \sum_{j=1}^{m_i} W_{ij} B_{23}(t_{ij}) A_i(t_{ij}) \tag{33}$$

$$\alpha_{i0} \hat{\mathbf{t}}_i \hat{\mathbf{t}}_{i+1} \sum_{j=1}^{m_i} w_{ij} B_{23}(t_{ij}) B_{33}(t_{ij}) - \alpha_{i1} \hat{\mathbf{t}}_{i+1}^2 \sum_{j=1}^{m_i} B_{33}^2(t_{ij}) = -\hat{\mathbf{t}}_{i+1} \sum_{j=1}^{m_i} w_{ij} B_{33}(t_{ij}) A_i(t_{ij}) \tag{34}$$

TABLE 1
ALGORITHM 1: Global Optimization Curve Fitting

Input: A set of data points to be fitted, corners and joint points of the data point set, an error margin *MaxPntErr,* and a maximum loop count *MaxNumLoop*
Output: Piecewise cubic Bézier curves specified by their control points

Begin
01. Break the data set into segments and sections based on given corner and joint points;
02. **For** each segment **Do**
03. **Begin**
04. Compute initial $\hat{\mathbf{t}}_i$ values for each corner and joint points (see [11]);
05. Compute initial t_{ij} values for all the data points (see [9]);
06. **Do**
07. Compute α_{i0} and α_{i1} for each Bézier curve piece Q_i from Eqs. (16), (17)
08. Compute t_{ij} values for each data points from Eq. (31)
09. Compute $\hat{\mathbf{t}}_i$ values for each corner and joint points using Eq. (28)
10. Compute maximum fitting error *FitErr* between $Ci(j)$ and Q_i;
11. Increase the number of iteration steps *NumLoop*;
12. **Until** (*FitErr* < *MaxPntErr*) or (*NumLoop* > *MaxNumLoop*);
13. **End;**
End.

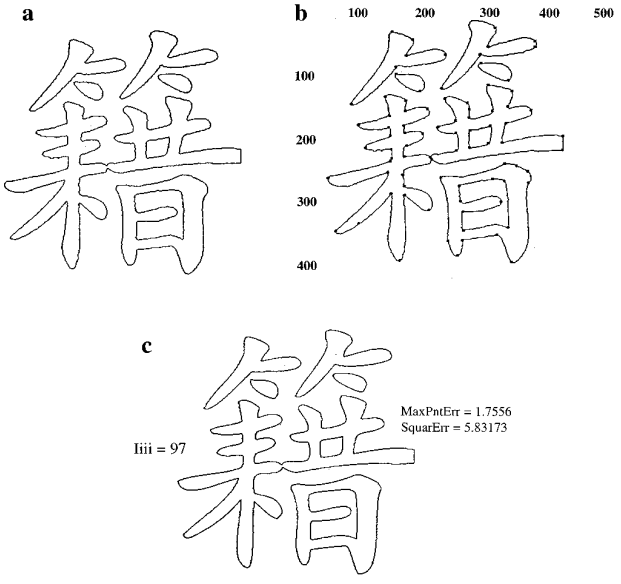

FIG. 8. An example showing the curve-fitting process.

$$i \in [1, n-1],$$

and the equations to compute $\hat{\mathbf{t}}_i$ will be modified as

$$\begin{aligned}\hat{\mathbf{t}}_1\alpha_{10}^2\sum_{j=1}^{m_i} w_{1j}B_{23}^2(t_{ij}) - \hat{\mathbf{t}}_2\alpha_{10}\alpha_{11}\sum_{j=0}^{m_i} w_{1j}B_{23}(t_{1j})B_{33}(t_{1j}) \\ = -\alpha_{10}\sum_{j=0}^{m_i} w_{1j}B_{23}(t_{1j})A_1(t_{1j})\end{aligned} \tag{35}$$

for $i = 1$;

$$\begin{aligned}\hat{\mathbf{t}}_{n-1}\alpha_{n-10}\alpha_{n-11}\sum_{j=1}^{m_{n-1}} w_{n-1j}B_{23}(t_{n-1j})B_{33}(t_{n-1j}) \\ - \hat{\mathbf{t}}_n\alpha_{n-11}^2\sum_{j=1}^{m_{n-1}} w_{n-1j}B_{33}(t_{n-1j}) \\ = -\alpha_{n-11}\sum_{j=1}^{m_{n-1}} w_{n-1j}B_{33}(t_{n-1j})A_{n-1}(t_{n-1j})\end{aligned} \tag{36}$$

for $i = n$; and

$$\begin{aligned}-\hat{\mathbf{t}}_{i-1}\alpha_{i-10}\alpha_{i-11}\sum_{j=1}^{m_{i-1}} w_{i-1j}B_{23}(t_{i-1j})B_{33}(t_{i-1j}) \\ + \hat{\mathbf{t}}_i\{\alpha_{i0}^2\sum_{j=1}^{m_i} w_{ij}B_{23}^2(t_{ij}) \\ \alpha_{i-11}^2\sum_{j=1}^{m_{i-1}} w_{i-1j}B_{23}^2(t_{i-1j})\} - \hat{\mathbf{t}}_{i+1}\alpha_{i0}\alpha_{i1}\sum_{j=0}^{m_i} w_{ij}B_{23}(t_{ij})B_{33}(t_{ij}) \\ = -\alpha_{i0}\sum_{j=1}^{m_i} w_{ij}B_{23}(t_{ij})A_i(t_{ij}) \\ + \alpha_{i-11}\sum_{j=0}^{m_{i-1}} w_{i-1j}B_{33}(t_{i-1j})A_{i-1}(t_{i-1j})\end{aligned} \tag{37}$$

for $i \in [2, n-1]$.

Since w values have already been computed during critical points detection process and are readily available, there is no additional overhead to apply weighted fit algorithm over the curve fit algorithm we introduced in last section. However, the improvement is quite obvious as can be seen in the example figures.

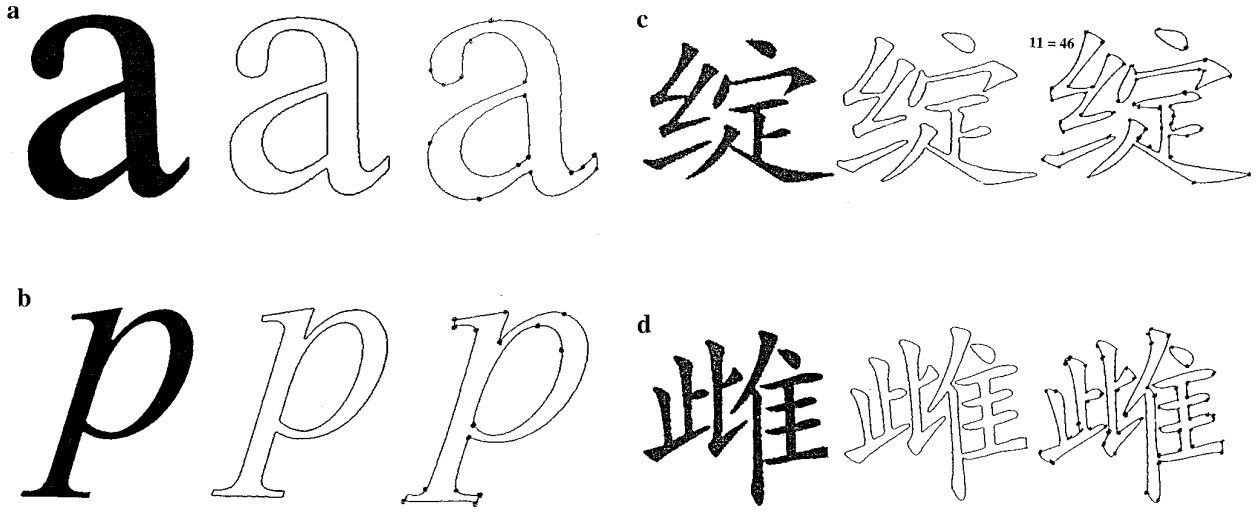

FIG. 9. Additional curve-fitting results.

6. EXPERIMENTAL RESULTS

The curve-fitting algorithm has been tested extensively on many font, logo, and other images. The result to fit some font images are given here. The original font images of art work were scanned into a computer at 300 DPI resolution. The standard image size was 1024×1024 pixels. The algorithm first extracted the boundaries from these images, then automatically fitted the boundaries and finally produced the Bézier curve descriptions of the images' outlines. The font images used for testing included those of English characters, numbers, Chinese characters, Kanji characters, Korean characters, etc., and they all gave very good results. In addition, the current recursive least square algorithm was reliable. No crash and divergence were found.

Some of the testing results are shown in Figs. 8 and 9. In each example, the first figure gives the original image, the second one presents the images' outlines and the last one shows the fitting result. The fitting results are good in both the quality and efficiency.

A comparison between some samples is given in Fig. 10. The *Fontographer,* a commercial program, is chosen to compare with our curve fitting algorithm. As can be seen, the difference is obvious. The current algorithm gave a much better result than the *Fontographer* did in terms of fitting accuracy and efficiency. On the average, our algorithm used 15% less Bézier curve pieces to fit the font's image boundaries and had much better fitting results. The characters generated by our algorithm are more accurate and smooth, which can be seen in Fig. 10.

7. CONCLUSIONS

A new curve-fitting algorithm is presented. The algorithm first identifies corner and joint points from a given set of data points. Then it applies a global least-squares optimization fit to find a best fit. It does so by breaking a nonlinear problem into two linear subproblems and uses a recursive fit to obtain the final solution. To further improve the fitting's accuracy, a weighted least-squares technique is used. The whole curve-fitting algorithm has been successfully implemented and has been incorporated into a Chinese font generation system *FontScript* [13]. The program is written in C language and runs on any IBM-PC compatible computer. The algorithm is quite reliable. We have never found any crash or divergence.

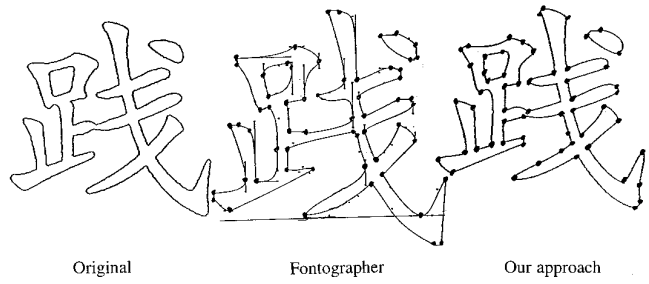

FIG. 10. A comparison of fitting results between our algorithm and the *Fontographer* program.

REFERENCES

1. Agfa, *Intellifont Sub-System Version 2.2,* AGFA, 1989.
2. D. Ansari and K.-W. Huang, Non-parametric dominant point detection, *Pattern Recognition* **24**(9), 1991, 849–862.
3. D. Ansari and E. J. Delp, On Detecting Dominant Points, *Pattern Recognition* **24**(5), 1991, 441–451.
4. M. J. Banks and E. Cohen, Real-time spline curves from interactively sketched data, *IEEE Comput. Graphics Appl.,* May 1992, 60–68.
5. L. Beus and S. S. H. Tiu, An improved corner detection algorithm based on chain-code plane curves, *Pattern Recognition* **20**(3), 1987, 291–296.
6. Freeman and L. S. Davis, A corner finding algorithm for chain-coded curves, *IEEE Trans. Comput.* **C-26,** 1977, 297–303.
7. H.-C. Liu and M. D. Srimath, Corner detection from chain-code, *Pattern Recognition* **3**(12), 1990, 51–68.
8. Pavlidis, Curve fitting with conic splines, *ACM Trans. Graphics* **2,** 1983, 1–31.
9. M. Plass and M. Stone, Curve-fitting with piecewise parametric cubics, *Comput. Graphics* **17,** July 1983, 229–239.
10. L. Piegl, Interactive data interpolation by rational Bézier curves, *IEEE Comput. Graphics Appl.,* Apr. 1987, 45–58.
11. P. J. Schneider, An algorithm for automatically fitting digitized curves, in *Graphics Gems* (Andrew S. Glassner, Ed.), pp. 612–625, Academic Press, San Diego, 1990.
12. L. Shao, C. L. Kai, and T. M. Shyan, On corner detection from chain-code, in *Proc. of 5th Int'l Symposium on IC Technology, Systems & Applications, Singapore, Sept. 15–17, 1993.*
13. L. Shao, H. Zhou, and C. K. Wah, A Chinese font generation system, *Int. J. Chinese Comput.,* Mar. 1995.
14. B. Kr. Ray and K. S. Ray, Detection of significant points and polygonal approximation of digitized curves, *Pattern Recognition Lett.* **13**(6), 1992, 443–452.
15. C. K. Wah, *Document Image Segmentation and Classification,* Master's Thesis, School of EEE, Nanyang Technology University, Singapore, Aug. 1995.