Pattern Recognition Letters 8 (1988) 237–249
North-Holland

November 1988

Binary contour coding using Bézier approximation

S.N. BISWAS, S.K. PAL and D. DUTTA MAJUMDER
Electronic and Communication Sciences Unit, Indian Statistical Institute, Calcutta 700 035, India

Received 16 January 1988
Revised 28 July 1988

Abstract: A method for coding of binary image contour using Bézier approximation is proposed. A set of key pixel (guiding pixels) on the contour is defined which enables the contour to be decomposed into arcs and straight line segments. A set of cleaning operations has been considered as an intermediate step before producing the final output.

The quality of faithful reproduction of the decoded version has been examined through the objective measures of shape compactness and the percentage error in area. Finally, the bit requirement and the compression ratios for different input images are compared with the existing ones.

Key words: Contour coding, Bézier approximation, Bresenham algorithm.

1. Introduction

Image coding is a technique which represents an image, or the information contained in it with fewer bits. Its objective is to compress the data for reducing its transmission and storage costs while preserving its information. Various techniques such as spatial domain methods, transform coding, hybrid coding, interframe coding etc. where both exact (error-free) and approximate (faithful replica) coding algorithms for binary and graytone image have been formulated are available in [1–8]. Approximate coding of graytone contour for its primitive (lines and arcs of different degree of curvature) extraction using fuzzy sets is described by Pal et al. [4–5].

Bézier approximation technique [6, 7] which uses Bernstein polynomials as the blending function provides a successful way to approximate an arc (not having any inflexion point) from a set of minimum three control points. The approximation scheme is simple and useful for its axis independence property. It is also found to be computationally efficient.

The present work attempts to formulate an algorithm for approximate coding of binary images based on Bézier approximation technique. A contour is first of all decomposed here into a set of arcs and line segments. For this, a set of key pixels are defined on the contour and the vertices of Bézier characteristic triangles corresponding to an arc are coded. Regeneration technique involves Bersenham's algorithm [8] in addition to the Bézier method. During the regeneration process, key pixels one considered to be the guiding pixels and their locations are therefore in no way disturbed. In order to preserve them, and to maintain the connectivity property some intermediate operations e.g., deletion and shifting of undesirable pixels generated by Bézier approximation, and insertion of new pixels are introduced in order to have better faithful reproduction.

Effectiveness of the algorithms is compared with two existing algorithms based on contour run length coding (CRLC) [9] and discrete line segment coding (DLSC) [10]. The compression ratios of the proposed methods are found to be significantly improved without affecting the quality much when a set of images is considered as input. The compactness and the difference in area between the input and output versions keeping the locations of the key pixels the same are also computed to provide a measure of the error.

2. Bézier approximation technique

Bernstein polynomials

The Bernstein polynomial approximation of degree m to an arbitrary function F: $[0,1] \to R$ is defined as

$$B_{im}[f(t)] = \sum_{i=0}^{m} f(i/m)\, \phi_{im}(t)$$

where the weighting functions ϕ_{im} are, for fixed t, the discrete binomial probability density functions for a fixed probability,

$$\phi_{im}(t) = \binom{m}{i} t^i (1-t)^{m-i}, \quad i = 0, 1, \ldots, m \qquad (1)$$

where

$$\binom{m}{i} = \frac{m!}{(m-i)!\,i!}.$$

The remarkable characteristics of the Bernstein polynomials are the extent to which they mimic the principal features of the primitive function f and the fact that the Bernstein approximation is always at least as smooth as the primitive function f where 'smooth' refers to the numbers of undulations, the total variation etc.

Bézier curves

This class of curves was first proposed by Bézier [6, 7]. The parametric form of the curves is

$$X = P_x(t), \qquad (2a)$$

$$Y = P_y(t). \qquad (2b)$$

Let (x_0, y_0), $(x_1, y_1), \ldots, (x_m, y_m)$ be $(m+1)$ ordered points in a plane. The Bézier curve associated with the polygon through the above points is the vector valued Bernstein polynomial and is given by

$$P_x(t) = \sum_{i=0}^{m} \phi_{im}(t)\, x_i, \qquad (3a)$$

$$P_y(t) = \sum_{i=0}^{m} \phi_{im}(t)\, y_i \qquad (3b)$$

where $\phi_{im}(t)$ are the binomial probability density functions of (1).

In the vector form, the equations (3) are

$$P(t) = \begin{pmatrix} P_x(t) \\ P_y(t) \end{pmatrix} \quad \text{and} \quad v_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

so that

$$P(t) = \sum_{i=0}^{m} \phi_{im}(t)\, v_i. \qquad (4)$$

The points $v_0, v_1, \ldots, v_m$ are known as the guiding points or the control points.

From equation (4) it is seen that

$$P(0) = v_0 \quad \text{and} \quad P(1) = v_m.$$

Thus the range of t significantly extends from 0 to 1. The derivative of $P(t)$ is

$$\begin{aligned} P'(t) = {} & -m(1-t)^{m-1} v_0 \\ & + \sum_{i=1}^{m-1} \binom{m}{i} \{ i\, t^{i-1} (1-t)^{m-i} \\ & - (m-i) t^i (1-t)^{m-i-1} \} v_i \\ & + m t^{m-1} v_m. \end{aligned}$$

Now $P'(0) = m(v_1 - v_0)$ and $P'(1) = m(v_m - v_{m-1})$. Thus the Taylor series expansion near zero is

$$\begin{aligned} P(t) &= P(0) + t\,P'(0) + \text{ higher order terms of } t \\ &\approx v_0(1 - mt) + \cdots \end{aligned}$$

and an expansion near one is

$$\begin{aligned} P(t) &= P(1) - (1-t)P'(1) + \text{higher order terms of } (1-t) \\ &\approx v_m\{1 - m(1-t)\} + m(1-t) v_{m-1}. \end{aligned}$$

It is now clear that as $t \to 0$ the Bézier polynomial lies on the line joining v_0 and v_1 and for $t \to 1$ on the line joining v_{m-1} and v_m. This means that these lines are tangents to the curve at v_0 and v_m.

Also since $\sum_{i=0}^{m} \phi_{im}(t) = 1$ the Bézier curve lies inside the convex hull of the control points.

For cubic Bézier curves, $m = 3$. The control polygon then consists of four control vertices v_0, v_1, v_2, v_3. The Bernstein polynomials for this case are

$$\begin{aligned}\phi_{0,3}(t) &= (1-t)^3 = -t^3 + 3t^2 - 3t + 1,\\ \phi_{1,3}(t) &= 3t(1-t)^2 = 3t^3 - 6t^2 + 3t,\\ \phi_{2,3}(t) &= 3t^2(1-t) = -3t^3 + 3t^2,\\ \phi_{3,3}(t) &= t^3,\end{aligned}$$

and the corresponding Bézier curve is

$$P(t) = (1-t)^3 v_0 + 3t(1-t)^2 v_1 + 3t^2(1-t)v_2 + t^3 v_3. \tag{5}$$

Though the cubic Bézier curve is widely used in computer graphics [11] we have used here its quadratic version to make the procedure faster enough.

For the quadratic Bézier curve, $m = 2$ and the control polygon always consists of three points. The Bernstein polynomials in this case are

$$\begin{aligned}\phi_{02}(t) &= (1-t)^2 = 1 - 2t + t^2,\\ \phi_{12}(t) &= 2(1-t)t = 2t - 2t^2,\\ \phi_{22}(t) &= t^2.\end{aligned}$$

Thus

$$P(t) = [\phi_{02}, \phi_{12}, \phi_{22}] \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = [t^2\ t\ 1]\,[c] \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}$$

where the coefficient matrix

$$[c] = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

In the polynomial form the Bézier curve is

$$P(t) = t^2(v_0 + v_2 - 2v_1) + t(2v_1 - 2v_0) + v_0. \tag{6}$$

This is a second degree polynomial and can be computed in a much faster way than in the Horner's process [11].

Bresenham algorithm

The underlying concept of the Bresenham algorithm for generating the points for a straight line segment restricted to an octant, given its two end points, lies in checking the proximity of the actual line to the grid location. Let (x_1,y_1) and (x_2,y_2) be the two points through which a discrete straight line segment is required. For this, the intercept of the line segment with the line at $x = x_1 + 1$, $x_1 + 2, \ldots, x_2$ are considered. If the intercept with the line at $x = x_1 + 1$ is closer to the line at $y = y_1 + 1$, then the point $(x_1 + 1, y_1 + 1)$ better approximates the line segment in question than the point $(x_1 + 1, y)$. This means if the intercept is greater than or equal to half the distance between $(x_1 + 1, y)$ and $(x_1 + 1, y_1 + 1)$ then the point $(x_1 + 1, y_1 + 1)$ is se-

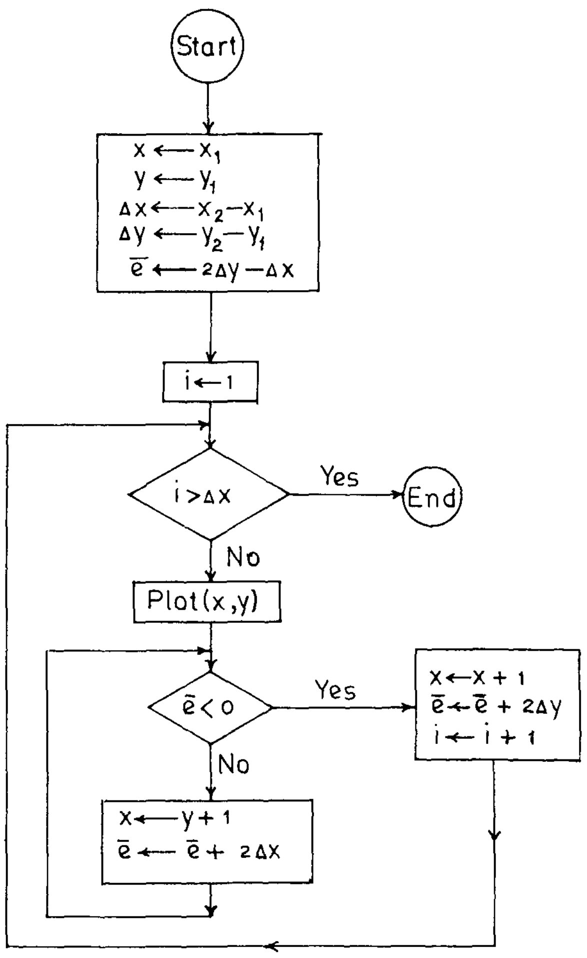

Figure 1. Flow chart for Bresenham's algorithm for generating straight line in first octant.

lected for approximation otherwise the point $(x_1 + 1, y)$ is selected. Next, the intercept of the line segment with the line at $x = x_1 + 2$ is considered and the same logic is applied for the selection of points.

Now, instead of finding the intercept an error term, e, is used for the selection purpose. Initially $e = -\frac{1}{2}$ and the initial point (x_1, y_1) is selected. The slope of the line, $\Delta y/\Delta x$ is added to e and the sign of the current value of $e = e + \Delta y/\Delta x$ is tested. If it is negative, then the point is selected along the horizontal line, i.e. x is incremented by one and y remains the same. The error term is then updated by adding the slope to it. But if the error term is positive (or two), then the point is selected along the vertical line, i.e. both x and y are incremented by one. The error term is updated by decreasing one from it.

For integer calculation, e is initialized to $\bar{e} = 2\Delta y - \Delta x$ because $2\Delta y - \Delta x = 2e\Delta x = \bar{e}$ (say). The details of the algorithm for the first octant is given in the flow-chart as shown in Figure 1.

3. Key pixels and contour approximation

A. Key pixels

In the analytic plane the contour of an object exhibits sharp maxima and minima and these points can be detected almost accurately without much difficulty. However when the contour is digitized in a two-dimensional array space of $M \times N$ points or pels or pixels, the sharpness in the curvature of the contour is destroyed due to the information loss inherent in the process of digitization. The error is known as the digitization error. Consequently it becomes rather difficult and complicated to estimate the points of maxima and minima. An approximate solution to this problem is to define a set of pixels, we call key pixels which are close to the points of maxima and minima.

For example, consider a function $f(x)$ in the discrete plane. When $f(x)$ is constant in an interval $[k_1, k_2]$, the corresponding analytic function $f_a(x)$ may exhibit local maxima and minima (or global maximum or minimum) anywhere within the interval as shown in the Figures 2(a) and 2(b).

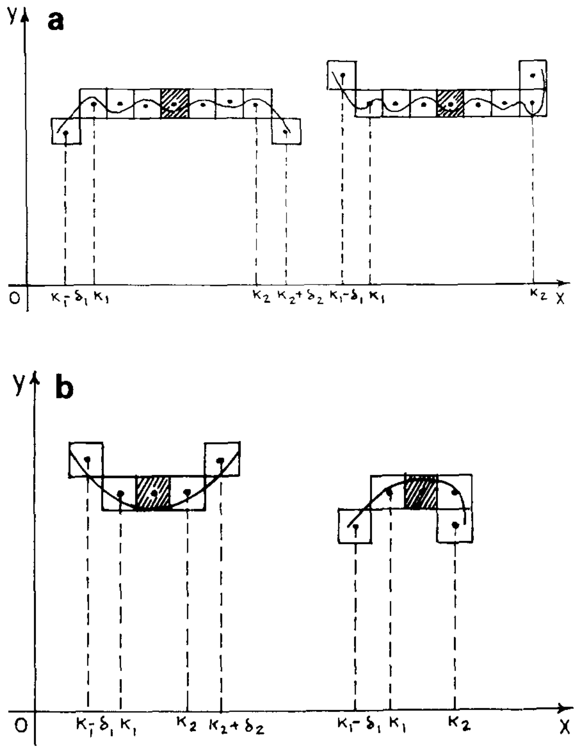

Figure 2. Possible behaviour of $f_a(x)$ when $f(x)$ is constant. (a) Considering local maxima/minima of $f_a(x)$. (b) Considering global maximum/minimum of $f_a(x)$. ■ denotes the position of the key pixel.

If we get a pixel either direct-connected or outward corner-connected to the end pixels of the interval $[k_1, k_2]$ such that both the values of $f(x)$ at these pixels are either greater or smaller than its value in the interval, then we assume a point of maximum or minimum to exist at the mid-point of the interval, i.e. at $x = (k_1 + k_2)/2$ if $(k_1 + k_2)$ is even and at $x = (k_1 + k_2 + 1)/2$ if $(k_1 + k_2)$ is odd. Let us consider this point or pixel in the discrete plane to be a key pixel. Another example for the existence of a key pixel shown by ■ is depicted in Figure 3 for which $f(x)$ is not constant over an interval.

B. Definition

A function $f(x)$, constant in $[k_1, k_2]$, in the discrete plane is said to have a key pixel P at $x = c$ (where $c = (k_1 + k_2)/2$ or $(k_1 + k_2 + 1)/2$ corresponding to even and odd values of $(k_1 + k_2)$) provided there exist $\delta_1, \delta_2 \in \{0, 1\}$ such that in both the intervals $[(k_1 - \delta_1), k_1]$ and $[k_2, (k_2 + \delta_2)]$

$$\text{either} \quad f(c) > f(x)$$
$$\text{or} \quad f(c) < f(x).$$

When $k_1 = k_2 = c$, the definition is applicable for Figure 3 where $\delta_1 = \delta_2 = 1$.

It is to be noted here that the above definition corresponds to the Figures 2 and 3 where key pixels lie on a horizontal sequence of pixels for the interval $[k_1, k_2]$ of x. Similarly, key pixels can also be defined for a vertical sequence of pixels for the interval $[k_1, k_2]$ of y.

C. *Contour approximation*

Let $k_1, k_2, \ldots, k_p$ be p key pixels on a contour. The segment (the Geometrical Entity, GE) between two key pixels can then be classified as either an arc or a straight line. If the distance of each pixel from the line joining the two key pixels is less than a prespecified value, δ, say, then the segment is considered to be a straight line (Figure 4(c)); otherwise it is an arc. The arc may again be of two types, with all the pixels either lying on both sides (Figure 4(a)) or lying on the same side (Figure 4(b)) of the line joining the key pixels. Let us denote the GE in Figure 4(c) by L (line) and that in Figure 4(b) by CC (curve). It is therefore seen that the GE in Figure 4(a) is nothing but a combination of two CC's meeting at a point Q (point of inflexion).

Therefore, the key pixels on the contour of a two-tone picture can be used to decompose the contour into two types of GE's, namely arc and line.

Let us now consider Figure 5 where the curve CC in Figure 4(b) is first of all enclosed within a right

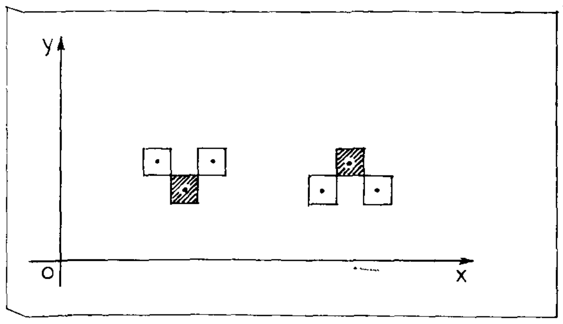

Figure 3. Position of the key pixel when $k_1 = k_2 = c$.

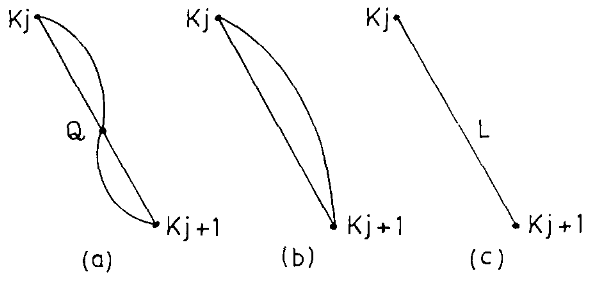

Figure 4. Types of GE. (a) Arc with inflexion point. (b) Arc. (c) Straight line.

triangle ABC where AC (the line joining k_j and k_{j+1}) is the hypotenuse and AB and BC are the horizontal and vertical lines respectively. It is proved in Appendix A that the arc CC will always be confined within a right triangle ABC. A line DF is then drawn which is parallel to the hypotenuse AC and passes through the pixel E of maximum displacement with respect to AC. Thus, the subtriangles ADE and CFE, so constructed, may be taken as the characteristic triangles to approximate the curve CC by quadratic Bézier approximation technique.

The preservation of the information of Bézier characteristic triangles with the help of key pixels forms the basis of the underlying concept of the proposed coding schemes.

4. Coding schemes

In the proposed method, two points (namely, E and C) are only stored to preserve the characteristic triangles corresponding to an arc when its starting

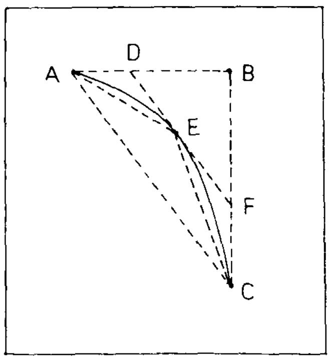

Figure 5. Bézier characteristic triangles for the arc AEC.

point A is known beforehand. The point D or F need not be stored because they can automatically be obtained from the aforesaid points. For example, D is the point of intersection of the horizontal line through A and the line through E and parallel to AC. It is to be noted here that the end point of GE is the starting point of its following GE.

Regarding straight lines, it is obvious that, only one point needs to be stored when the starting point is known.

The algorithm for key pixel extraction is shown in Appendix B.

Bit requirement

Let there be p different contours in a binary image of size $M \times N$ where $M = 2^m$ and $N = 2^n$. The contours may be of two types; either closed or open. If n_k and n_i are respectively the number of key pixels (including the end pixels for open contour) and points of inflexion on a contour, then the number of arcs and straight lines (segments) is $(n_k + n_i - 1)$. Of them, let n_s be the number of straight line segments. For closed contours the initial key pixel is the same as the final key pixel.

The codeword, s, of a GE is variable in length. s consists of two subwords s_1 and s_2. s_1 always represents identity (arc or line) of the GE while s_2 denotes its description. When the GE is an arc, s_2 gives the vertices of the characteristic triangle (for example, A,E,C in Figure 5). For a straight line segment, s_2 indicates the end points of the line segment. It is obvious that the current end point is always the starting point of the succeeding GE. The bit pattern representing a contour is therefore as displayed in Figure 6.

$\longleftarrow r_1 \longrightarrow$ $\longleftarrow r_2 \longrightarrow$ $\longleftarrow r_3 \longrightarrow$
[Type of contour] [Number of GE] [Starting point]

$\longleftarrow s_1 \longrightarrow$ $\longleftarrow s_2 \longrightarrow$ ....
[Identity of GE] [Description of GE]

$\longleftarrow s_1 \longrightarrow$ $\longleftarrow s_2 \longrightarrow$
[Identity of GE] [Description of GE]

Figure 6. Bit pattern.

Types of contours (open or closed) can be represented by a single bit.

In the worst case, all the GE's may be straight line segments and the number of key pixels may be MN. Thus it needs $(m + n)$ bits to represent the total number of GE's in a contour.

Identity of GE (arc or line) can be represented by a single bit.

Given a starting point of the contour, we need two points for describing an arc and one point for a straight line. Each point can be represented by $(m + n)$ bits.

Therefore, for describing an open contour consisting of $((n_k + n_i - 1) - n_s)$ arcs and n_s lines we need

$$T_o = (m + n) + 2(m + n)((n_k + n_i - 1) - n_s) + (m + n)n_s$$

bits where the first term corresponds to the starting point. For a closed contour, the amount of bits required is $T_c = T_o - (m + n)$ since the last key pixel (end point) corresponds to the starting point.

From Figure 6, it is therefore seen that T_o (or T_c) gives the bit requirement for s_2's only. The total requirement considering the remaining entities of Figure 6, will therefore be

$$B_{\text{total}} = \alpha + \beta + \gamma + \delta$$

where

$\alpha =$ requirement corresponding to $r_1 = 1$,
$\beta =$ requirement corresponding to $r_2 = (m + n)$,
$\gamma =$ requirement corresponding to s_1's $= (n_k + n_i - 1)$,
$\delta = T_o$ or T_c.

5. Decoding

The coded binary string output corresponding to the method is shown in Figure 7. $(m + n)$ indicates the word length for the number of GE's whereas $(m) + (n)$ denote the co-ordinates of a point.

Decoding of the string of Figure 7 is based on the following notations.

The first bit (l_1) indicates the type of contour (i.e. $l_1 = 0$ for open and 1 for closed). The next sequence

$$\begin{array}{ccccc}
l_1 & l_2 & l_3 & l_4 & l_5 \\
(*) & (***\cdots*) & (***\cdots*) & (***\cdots*) & (*) \\
 & (m+n) & (m)+(n) & (m)+(n) & \\
l_6 & l_4 & l_5 & l_6 & \\
(***\cdots*) & (***\cdots*) & (*) & \underbrace{\qquad} & \\
(m)+(n) & (m)+(n) & & & \\
\cdots\cdots & l_4 & l_5 & l_6 & \\
 & (***\cdots*) & (*) & \underbrace{\qquad} & \\
 & (m)+(n) & & &
\end{array}$$

Figure 7. Coded binary string output.

l_2 of $(m+n)$ bits represents the number of GE's present in the contour. The first m bits of the sequence l_3 denote the value of the ordinate whereas the remaining n bits give the value of the abscissa of the starting point. Similarly, the co-ordinates of the first key pixel is given by the sequence l_4. Bit l_5 says whether the GE between the points represented by l_3 and l_4 is an arc or a straight line. $l_5 = 0$ for line and 1 for arc. If there is an arc, then the following sequence l_6 is considered to indicate the point E (as in Figure 5); otherwise, the sequence l_6 will be absent.

As soon as an arc or line is reconstructed, the preceding key pixel point becomes the new starting point.

The point designated by the following sequence l_4 then represents the new key pixel for further reconstruction.

The procedure for decoding continues until the number of GE's, as represented by the sequence l_2, is exhausted. After that, a new contour is started with the first bit as l_1.

6. Regeneration technique

During the decoding procedure, if the GE between two key pixels is found to be a straight line, then it is generated by the Bresenham algorithm as mentioned in Section 2. If the GE is an arc, the Bézier characteristic triangles are first of all constructed in order to generate its quadratic approximated version.

Recursive computation algorithm

The algorithm for computing the values of 2nd order Bézier approximation curve using a forward difference scheme is described below. Let

$$y = at^2 + bt + c$$

be a polynomial representation of the equation (6) where the constant parameters a, b and c are determined by the vertices of the Bézier characteristic triangle.

Suppose, a number of points (values of y) on the arc are to be evaluated for equispaced value of the independent variable t.

The usual Newton's method of evaluating the polynomial results in multiplications and does not make use of the previously computed values to compute new values.

Assume that the parameter t ranges from 0 to 1. Let the incremental value be q. Then the corresponding y values will be c, $aq^2 + bq + c$, $4aq^2 + 2bq + c$, $9aq^2 + 3bq + c, \ldots$. Let us now form the difference table (see Table 1). Observe that

$$\Delta^2 y_j = 2aq^2$$

and

$$y_{j+2} - 2y_{j+1} + y_j = 2aq^2 \qquad \text{for all } j \geq 0.$$

This leads to the recurrence formula $y_2 = 2y_1 - y_0 + 2aq^2$ that involves just three additions to get the next value from the two preceding values at hand. Thus one does not need to store all the points on the curve.

Table 1
Difference table for recursive computation of points of Bézier curve

t	y	Δy	$\Delta^2 y$
0	c	$aq^2 + bq$	$2aq^2$
q	$aq^2 + bq + c$	$3aq^2 + bq$	$2aq^2$
$2q$	$4aq^2 + 2bq + c$	$5aq^2 + bq$	$2aq^2$
$3q$	$9aq^2 + 3bq + c$	$7aq^2 + bq$	
$4q$	$16aq^2 + 4bq + c$		

7. Implementation strategies

After coding a single pixel width contour input, the regeneration algorithm as described before is used to decode and result in its approximated version (output). During regeneration, the outer contour is only traced using Freeman's chain code (clockwise sense) assuring the positions of key pixels on it. In other words, key pixels are considered to be the guiding pixels (being important for preserving the input shape) during reproduction.

It is to be noted that due to the approximation scheme, sometimes the following undesirable situations may arise:

(1) The regenerated contour may not have single pixel width.

(2) The key pixel may become an interior pixel of the contour.

To overcome these situations we trace the contours from the ordered regenerated data set, considering the following operations.

A. Deletion of pixels

During the contour tracing if a pixel on the contour finds more than one neighbour in its 8-neighbourhood domain, then the exterior pixel on the contour is kept while deleting the rest (pixels on the interior contour). But if there is a key pixel falling in such neighbourhood, then the key pixel is retained as the contour pixel and the rest are deleted. This enables us to keep the key pixel always on the contour, thus making the approximation of the input better. Figures 8(a) and (b) depict the situations. Considering 'c' to be the current pixel and 'p' the previous pixel, the contour (clockwise) is 'a' for a situation as shown in Figure 8(a) but if the situation is as in Figure 8(b) the next pixel on the contour is then k (the key pixel).

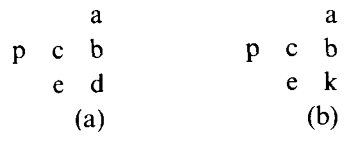

Figure 8. Deletion of pixels: (a) in absence of key pixel, (b) in presence of key pixel.

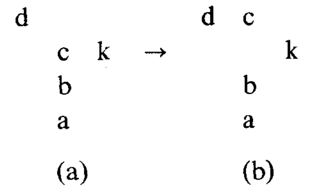

Figure 9. Shifting of pixels: (a) contour before shifting, (b) contour after shifting.

B. Shifting of pixels

Suppose a GE is generated and a key pixel is reached. Now during the generation of a following GE, its first data point may make the preceding key pixel lie on the interior contour. For example, consider Figure 9(a). Here a b k is part of the GE which is already generated. Now generating the next GE: k c d,..., the first move from k to c makes the key pixel (k) lie on an interior contour.

In such cases, the data point c is shifted as shown in Figure 9(b). This preserves connectedness of the pixel c with both the GE's and also ensures single pixel width of the contour.

C. Undesirable loop

Sometimes in the vicinity of key pixels an undesirable loop (contour with a single pixel hole) may appear due to the approximated generation procedure. For example consider Figure 10. Here GE's a k_1 k_2 k_3 are already generated. The next move from k_3 to b creates an undesirable loop having single pixel hole.

To overcome this situation, the pixel b is shifted along with an insertion of a new pixel e (as shown

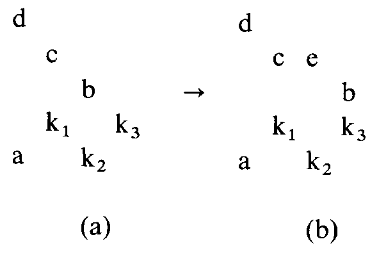

Figure 10. Undesirable loop: (a) before cleaning, (b) after cleaning.

in Figure 10(b)). Since the shifting of b alone loses the connectivity property between k_3 and the subsequent pixels, it necessitates an insertion of a new pixel whose location is governed by the concept of minimum connected path.

8. Results and discussion

Figures 11(a), 12(a) and 13 show the digital contours of three different figures, namely, butterfly, chromosome and numeral-eight, which were con-

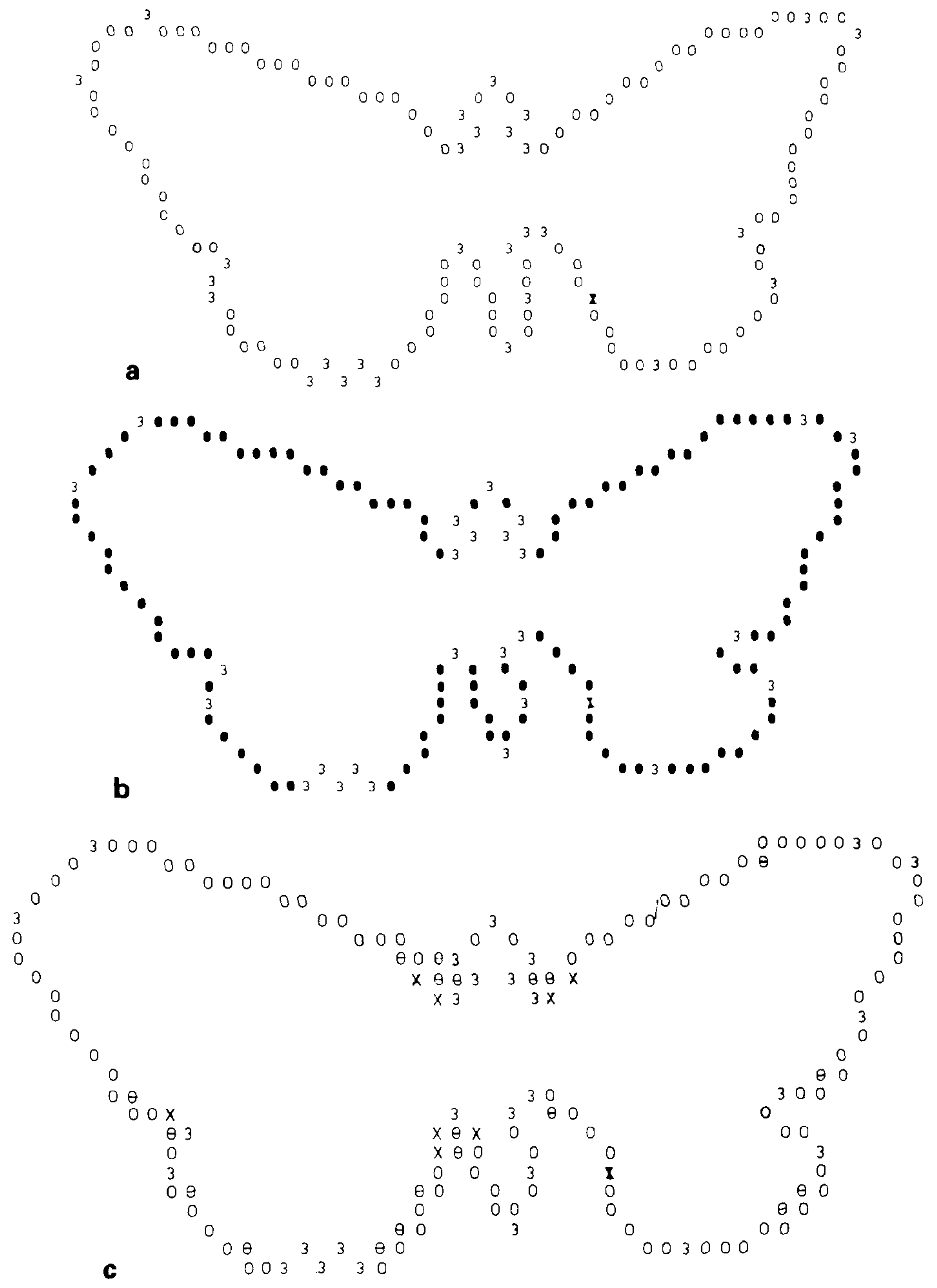
Figure 11. (a) Butterfly input. (b) Regenerated version. (c) Regenerated version before cleaning.

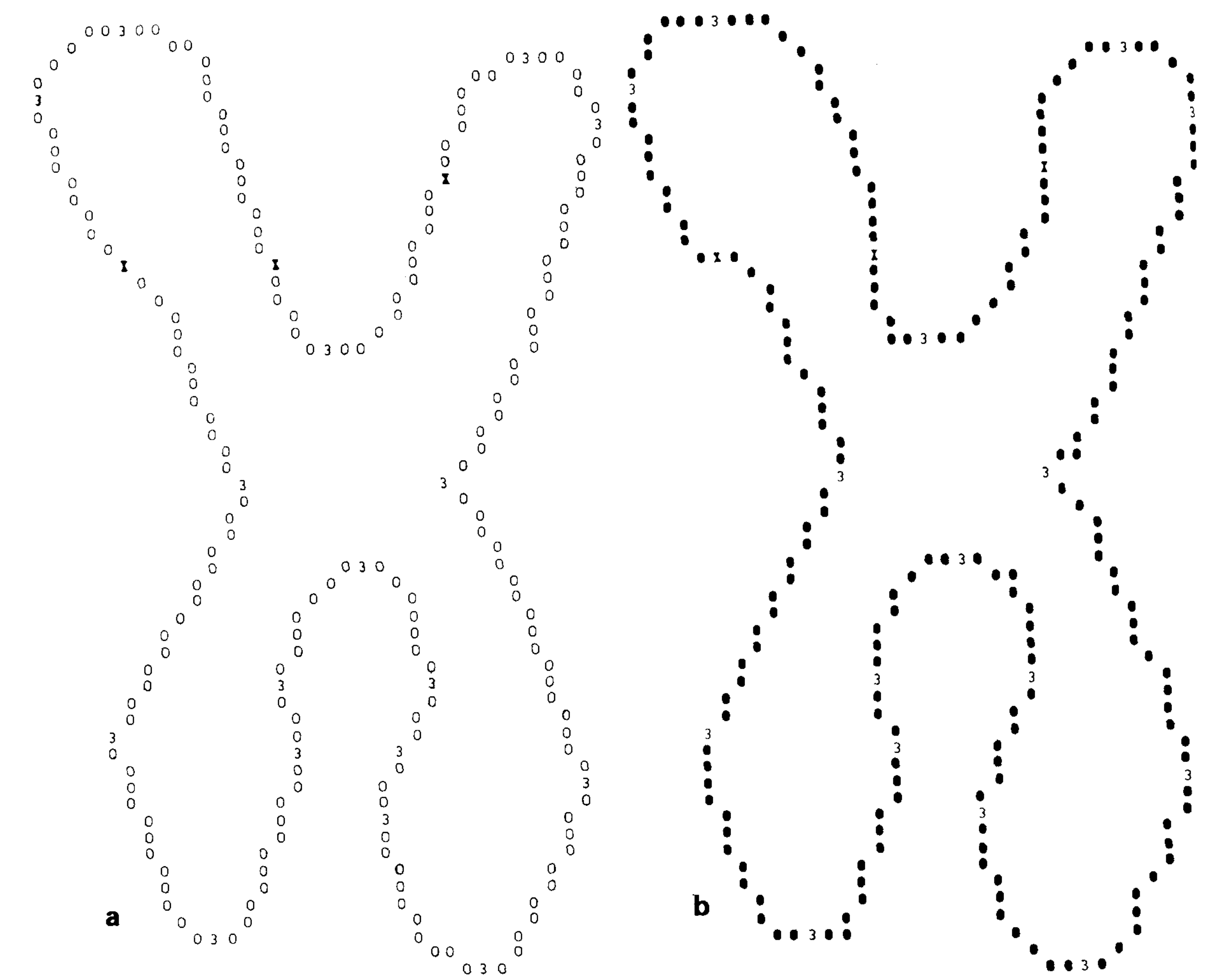

Figure 12. (a) Chromosome input. (b) Regenerated version.

sidered as input to the proposed coding schemes. The key pixels and the points of inflexion as detected on the input patterns are marked by '3' and 'X' respectively.

In Figure 13, the input image contour is represented by the pixels marked '5' and '0' along with '3' denoting its key pixels. The output corresponding to the butterfly and chromosome images are shown in Figures 11(b) and 12(b) respectively. The output version corresponding to the numeral-8 contour is marked by '5' and '●' in Figure 13 superimposing on its input. This superimposed diagram facilitates one to examine the visual proximity between the input and output versions. It is to be noted in this connection that the positions of key pixels in both input and output remain unaltered.

For coding the input patterns, the numbers of GE's in the butterfly, chromosome and numeral-8 were found to be 27, 19 and 16 respectively. Out of these figures the numbers of arcs were 9, 16 and 16. The contour of the numeral-8, as expected, has the minimum number of GE's and has no straight line.

As a typical illustration, the effectiveness of the cleaning operations (Section 7) performed on the generated points is demonstrated only for the butterfly image. Figure 11(c) shows such an intermediate state beforing producing its final decoded output. Here, θ denotes a pixel deleted and X corresponds to the position where a pixel is inserted to keep connectivity.

In order to study the efficiency of the coding schemes, the bit requirements for different input im-

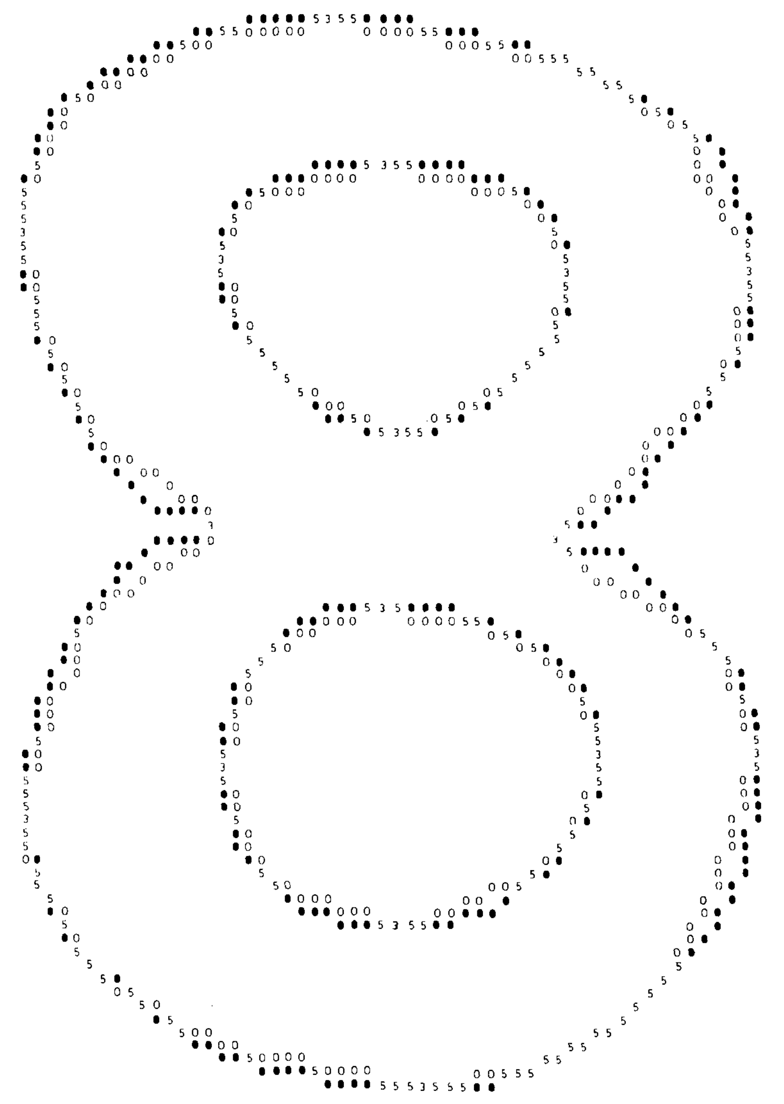

Figure 13. Numeral-8 input and its regenerated output together.

ages and their relative compression ratios are compared with those in the CRLC and DLSC methods. It is shown in Table 2 that in the proposed technique, the bit requirements are significantly less. It is obvious that the numeral-8 image consisting of few large arcs only provided higher compression ratio. The butterfly image on the other hand, has the largest number of GE's and thus provides lowest compression ratio.

As the coding schemes are approximate the regenerated image deviates from its original version. To observe the deviation of regenerated image quality through an objective measure we have calculat-

Table 2
Bit requirement

Figure	Bit requirement			δ_{rel} % (rel. CRLC)	δ_{rel} % (rel. DLSC)
	CRLC	DLSC	Proposed method		
Butterfly	799	531	435	183.67	122.06
Chromosome	1175	603	452	259.95	133.40
Numeral-8	2085	1169	474	439.87	246.62

Table 3
Error in regeneration

Figure	% Error in area	Compactness	
	Proposed method	Original figure	Generated figure
Butterfly	8.63	0.024635	0.025393
Chromosome	6.8	0.016061	0.016672
Numeral-8	2.92	0.014728	0.014589

ed the error in area and the shape compactness. For the calculation of the area and the perimeter of the contour we have used the technique proposed by Kulpa [12]. Since the key pixels are always on the contour and the generated arcs are between them and restricted by the respective Bézier characteristic triangles, the maximum error for an arc is the area of its pair of Bézier characteristic triangles. Also, for the above constraint the shape compactness can provide a good measure of the distortion introduced into the decoded images. Table 3 shows both the percentage error and the compactness of figures. It is thus seen that the decoded image in each case is a faithful reproduction of its input version. Here too, the butterfly/numeral-8 contour having the largest/smallest number of GE's incurred highest/lowest % error in their regeneration.

Finally, it is to be mentioned here that since the regeneration procedure uses the quadratic Bézier approximation technique, the decoded image display is very fast.

Acknowledgment

The authors would like to thank Mr. J. Gupta and S. Chakraborty for rendering the secretarial help.

References

[1] *Proc. IEEE* (1980). Special issue on Computer Graphics, Vol. 68.

[2] Ekstrom, M.P., Ed. (1984). *Digital Image Processing Techniques*. Academic Press, New York.

[3] Huang, T.S. (1977). Coding of two-tone images. *IEEE Trans. Com.* 25, 1406–1424.

[4] Pal, S.K., R.A. King and A.A. Hashim (1983). Image description and primitive extraction using fuzzy sets. *IEEE Trans. Syst. Man Cybernet.* 13 (1), 94–100.

[5] Pal, S.K. and D. Dutta Majumder (1986). *Fuzzy Mathematical Approach to Pattern Recognition*. Wiley, New York.

[6] Bézier, P.E. (1974). Mathematical and practical possibilities of UNISURF. In: Barnhill, R.E. and R.F. Riesenfeld, Eds., *Computer Aided Geometric Design*. Academic Press, New York.

[7] Barskey, B.A. (1983). A description and evaluation of various 3-D models. In: Kunii, T.L., Ed., *Computer Graphics (theory and applications)*. Springer, Berlin.

[8] Bresenham, J.E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems J.* 4 (1), 25–30.

[9] Morrin, T.H. (1976). Chain link compression of arbitrary black white images. *Computer Graphics and Image Processing* 5, 172–189.

[10] Chaudhuri, B.B. and M.K. Kundu (1984). Digital line segment coding: A new efficient contour coding scheme. *IEE Proc.* 131 (4), Pt.E, 143–147.

[11] Pavlidis, T. (1982). *Algorithms for Graphics and Image Processing*. Springer, Berlin.

[12] Kulpa, Z. (1977). Area and perimeter measurement of blobs in discrete binary pictures. *Computer Graphics and Image Processing* 6, 434–451.

Appendix A

Proposition 1. *In the discrete plane all the pixels on an arc between two key pixels remain always on or inside a right triangle with the line joining the key pixels as the hypotenuse. The other two sides of the right triangle are the horizontal and the vertical lines through the key pixels.*

Proof. When the key pixel is on the horizontal line at $x = c$ it follows from the definition of key pixel, that

$$\text{either} \quad f(c) > f(x)$$
$$\text{or} \quad f(c) < f(x)$$

in both the intervals $[k_1 - \delta_1, k_1]$ and $[k_2, k_2 + \delta_2]$ where $f(x)$ is constant in $[k_1, k_2]$ and $\delta_1, \delta_2 \in \{0, 1\}$. Thus

(1) Pixels at k_1 and k_2 are either corner connected or direct connected or its combination to the neighbouring pixels outside the interval $[k_1, k_2]$.

(2) When $k_1 = k_2 = c$, the key pixel will have at least one corner connection to its neighbouring pixels.

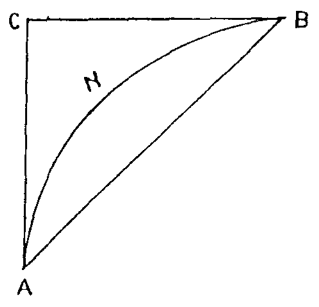

Figure A1. An arc with its associated right triangle.

Similar arguments hold when the key pixel lies on a vertical line.

Let ANB be the arc with A and B being two successive key pixels as shown in Figure A1. Now a pixel on the arc can go outside the line AC or BC if and only if (1) there exists a sequence of collinear pixels such that its end pixels are either corner connected or direct connected or its combination, or (2) there exists a pixel which has at least one corner connection with its neighbouring pixel.

Both these conditions lead to the existence of another key pixel outside the line AC or BC. This is a contradiction.

Appendix B

Algorithm for extraction of key pixels

$\{P_i\}_{i=1}^{n}$ are the contour points in the binary image and $\{(x_i, y_i)\}_{i=1}^{n}$ are their position co-ordinates. Since, for a closed contour, there is a possibility of missing the first key pixel we need to examine a few more points after the starting point is reached so as to enable one to get the same back.

4	3	2
5	●	1
6	7	8

Figure B1. Directional codes with respect to ●.

Step 1. Set $i \leftarrow 1$, count $\leftarrow 1$.

Find the initial direction code between P_i and P_{i+1} according to Figure B1. Let it be d_1.

Step 2. Increment $i \leftarrow i + 1$; if $i = n$ go to Step 7; otherwise find the directional code between P_i and P_{i+1}. Let it be d_2.

Step 3. If $d_1 = d_2$ go to Step 2; otherwise if d_1 div $2 = 0$ and d_2 div $2 = 0$ or if $|d_1 - d_2| = 3$ or 5 then return (x_i, y_i).

Step 4. Increment $i \leftarrow i + 1$; if $i = n$ go to Step 7; otherwise find the direction code between P_i and P_{i+1}. Let it be d_3.

Step 5. If $d_3 = d_2$ then count $\leftarrow$ count $+ 1$ and go to Step 4; otherwise if $|d_1 - d_3| = 0$ or 1 then set count $\leftarrow 1$, $d_1 \leftarrow d_3$ and go to Step 2 else do Step 6.

Step 6. If count div $2 = 0$ then return $(x_{i-\text{count}/2}, y_{i-\text{count}/2})$otherwisereturn$(x_{i-\text{count div } 2}, y_{i-\text{count div } 2})$.

Step 7. Stop.