# Contour coding through stretching of discrete circular arcs by affine transformation

Sambhunath Biswas*

*Machine Intelligence Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India*

Received 13 May 1998; received in revised form 6 October 1999; accepted 6 October 1999

**Abstract**

This paper presents a method for coding binary contour images. Coding is lossy in nature. A contour is first decomposed into line and arc segments through an extraction of some of its dominant pixels (called the key pixels including the pixels of inflexion). Detection of inflexion points (pixels) is made through the Gaussian circle and its image. Each contour arc is then approximated by suitably stretching a locally defined discrete circular arc through affine transformation(s). Stretching can be performed iteratively. To get discrete circular arcs, a generation algorithm has been presented along with their properties in conjunction with rings and discs. Performance of the coding algorithm has been compared to that of a method which reconstructs the contour image approximately. Comparison with the performance of exact or lossless methods has also been carried out to show how far it is away from them.

*Keywords:* Contour; Gaussian circle; Line; Arc; Affine; Stretching; Coding; Comparison

## 1. Introduction

One of the techniques in gray image coding is to consider the gray part of the homogeneous regions and their boundaries or contours. These contours are essentially binary in nature. The compression of contours in segmentation based coding plays a significant role and needs, therefore, an extensive investigation. Some of the works, in this area, can be found in [1–6]. The proposed method reconstructs a contour approximately with good fidelity. Methods reported in [1,5] also approximately reconstruct a contour, while others cited above reconstruct a contour exactly. Recently, Ghorbel et al. [7] has described a contour coding scheme in the context of motion estimation. The coding is based on the computation of invariant shape features. The invariant descriptors corresponding to shape features are used for reconstruction at the receiving end. Some related works can also be found in [8]. In this paper, we have checked the possibility of using discrete circular arcs under affine transformation to approximate and encode binary contour data. The developed approximation scheme uses affine transformations iteratively to judiciously approximate a contour segment. Since the approximation uses discrete circular arcs, we have also developed a discrete circle (d.c.) generation scheme and studied their properties concisely in a discrete array space of pels or points. The subject called digital or discrete geometry, is relatively new and the reported result in 1- and 2-dimensional digital geometry concern mainly straight line and circle. Of them, the conditions of digital straight line (d.s.l.) are given by Freeman [9] and Rosenfeld [10]. Wu [11] presents algorithms to recognize a string as that of a d.s.l. Generation of a d.s.l. given its slope is also reported [12]. Some interesting applications of d.s.l. in image processing problems are given in [3,13].

Similarly, the generation of a digital circle (d.c.) using Freeman's raster intersection digitization scheme has been reported [14]. The properties of the d.c., generated by this scheme have been studied [15,16]. It has been shown that the rings and discs formed by successive

generation of circles under this scheme contain holes or gaps, which are to be filled subsequently. The generation of d.c. using incremental steps along fixed coordinate axes is also reported [17].

The d.c. generation scheme that we have developed is different in nature because here the problem of holes in rings and discs are avoided. The error in the present d.c. approximation is comparable to one of the reported schemes [16] and the implementation algorithm is simple and straightforward. The d.c. generation scheme is of great use in binary contour coding. The coding method is found to be efficient. A comparison between the proposed technique and that in [1] shows that the proposed technique is superior. Comparison between the proposed method and exact methods in [2,3] shows how far the proposed approximate method is away from these two exact methods.

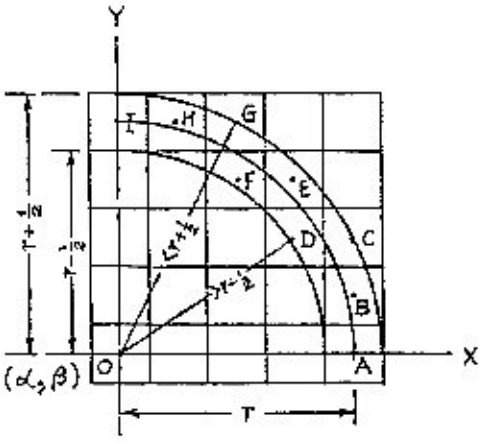



Fig. 1. One quadrant of a discrete circle for radius = 4.

## 2. Discrete circle, ring, disc and their properties

Consider a 2-dimensional discrete array space of $m \times n$ points or pixels or pels so that any point or pel $(x, y)$, $0 \leqslant x \leqslant m-1$, $0 \leqslant y \leqslant n-1$. $x, y, m, n \in I$ (set of integers) can be mapped to the continuous real plane by a unit square about the center point $(x \pm \frac{1}{2}, y \pm \frac{1}{2})$. Also, for simplicity and convenience, let the radii of the discrete circle, ring and disc be integer valued with center of the unit squares as stated above.

*Discrete circle (d.c.)*

A d.c. is a discrete space approximation to the circle defined in Euclidean geometry. In the present scheme of generation, a d.c. is defined as follows.

**Definition 1.** A d.c. with radius $r$ and center $(\alpha, \beta)$ is a set $S_r$ of 8-connected pels so that each pel $(x, y)$ satisfies the inequality [18]

$$r - \tfrac{1}{2} < |\sqrt{(x-\alpha)^2 + (y-\beta)^2}| < r + \tfrac{1}{2}. \quad (1)$$

Part of a d.c. with $r = 4$ is shown in Fig. 1 where each square with a dot inside is a member of $S_r$. The members are depicted as A, B, C, D, E, F, G, H and I for convenience.

Uniqueness of a d.c. under the above definition may be established by the following proposition.

**Proposition 1.** *For concentric $S_r$ and $S_t$*

$$S_r \cap S_t = \emptyset, \quad r \neq t. \quad (2)$$

**Proof.** Let there exist a pel $Q(x_1, y_1)$ so that $Q \in S_r$ and $Q \in S_t$, $r \neq t$. Then, from Definition 1,

$$r - \tfrac{1}{2} < |\sqrt{(x_1-\alpha)^2 + (y_1-\beta)^2}| < r + \tfrac{1}{2}$$

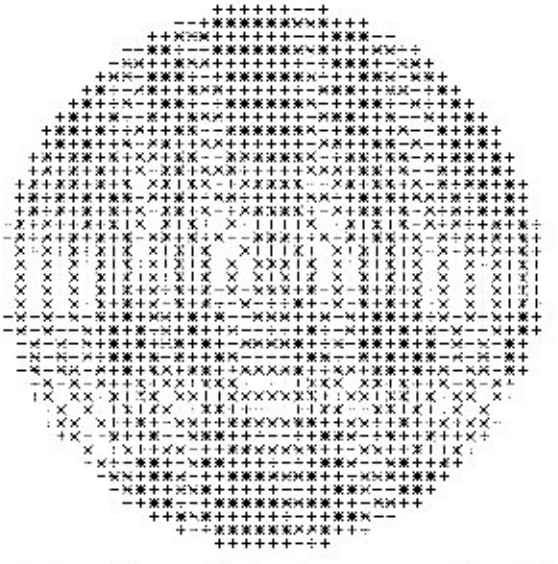


Fig. 2. A set of discrete circles (* and + represent circles of odd and even integer radii, respectively).

and

$$t - \tfrac{1}{2} < |\sqrt{(x_1-\alpha)^2 + (y_1-\beta)^2}| < t + \tfrac{1}{2}$$

since $r$ and $t$ are integers and $r \neq t$, both inequalities cannot be satisfied simultaneously by $(x_1, y_1)$. Hence all pels belonging to a d.c. are unique.

A set of d.c. generated by Definition 1 and the algorithms given in Section 3 is shown in Fig. 2. It should be noted that the d.c. generated by Definition 1 is the same as the d.c. generated by the scheme given in [14] for

some values of $r$. For $r \leqslant 50$ these values are 2, 3, 5, 7, 10, 12, 15 and 20.

### 2.1. Error in d.c. approximation

The important parameters of a discrete circle are its area and perimeter which infer how close a d.c. is to a circle in the Euclidean geometry. For perimeter measurement a subset $S'_r$ of $S_r$ which denotes 8-connected outer border of the d.c. is only considered. The pels belonging to $S_r - S'_r$ are interior pels. For example in Fig. 1, A, B, C, E, G, H and I are 8-connected outer border pels and D, F are interior pels. The area and perimeter were measured with Kulpa's method [16] and their expressions are given below.

Let $h$ and $d$ represent the number of non-diagonal and diagonal links in $S'_r$, respectively. Then the length of the border (perimeter) is given by

$$l_c = \frac{\pi(1+\sqrt{2})(h+\sqrt{d})}{8}. \tag{3}$$

The percentage error in contour length over the ideal one is

$$\varepsilon_p = \frac{l_c - l}{l} \times 100 \tag{4}$$

where $l = 2\pi r$. Similarly, the area of the d.c. is given by

$$A_c = N - \left(\frac{L}{2} + 1\right) \tag{5}$$

where $N$ is the number of pels in a disc of radius $r$ and $L = h + d$. Eqs. (3) and (5) do not represent the actual length and area. In Eq. (3), a correction factor of $\pi(1+\sqrt{2})/8$ and in Eq. (5), a correction term of $(L/2+1)$ are used to compute the respective parameters with good accuracy [16,19]. The percentage error in d.c. area over the ideal one is given by

$$\varepsilon_a = \frac{A_c - A}{A} \times 100 \tag{6}$$

where $A = \pi r^2$. Eqs. (4) and (6) are plotted against $r$ in Fig. 3 for the present method of d.c. generation as well as for the method given in [14]. It is seen that the error is comparable in the two methods.

*Rings and Discs*

Since a pel covers a square area in real space, a d.c. has some width in real space. A ring or a disc can therefore be generated by the union of circles of radii $r$, $r+1, \ldots, \ldots, r+m$. However it is interesting to observe the following properties in connection with the generation of a ring and a disc by the present method.

**Proposition 2.** *There cannot exist any gap or hole between any two concentric d.c.s of radii $r$ and $r+1$.*

**Proof.** According to Definition 1, the Euclidean distance $d$ between any pel of $s_r$ and the center of the d.c. satisfies the inequality

$$r - \tfrac{1}{2} < d < r + \tfrac{1}{2} \tag{7}$$

and for a concentric d.c. of radius $r+1$, this distance, say $d_1$ satisfies

$$r + 1 - \tfrac{1}{2} < d_1 < r + 1 + \tfrac{1}{2}, \tag{8}$$

i.e.,

$$r + \tfrac{1}{2} < d_1 < r + \tfrac{3}{2}.$$

Now let a pel $H(x_1, y_1)$ exist between the two d.c.s so that $H$ does not satisfy either Eq. (7) or Eq. (8) and hence it is a hole or a gap. Then the distance $d_g$ between $H$ and the center of the d.c.s should not belong to either of the open interval $(r-\frac{1}{2}, r+\frac{1}{2})$ and $(r+\frac{1}{2}, r+\frac{3}{2})$.

The only possibility then is $d_g = r + \frac{1}{2}$. Now $d_g^2 = (x_1 - \alpha)^2 + (y_1 - \beta)^2$, where $(\alpha, \beta)$ is the center of the d.c.s. The right-hand side is an integer since all its entries are integers. On the other hand, the left-hand side is not an integer since $d_g = r + \frac{1}{2}$. The possibility $d_g = r + \frac{1}{2}$ is therefore rejected and $H$ cannot exist.

**Definition 2.** A discrete ring (d.r.) with integer radius $r_1$ and $r_2, r_2 > r_1$ and integer center $(\alpha, \beta)$ is given by

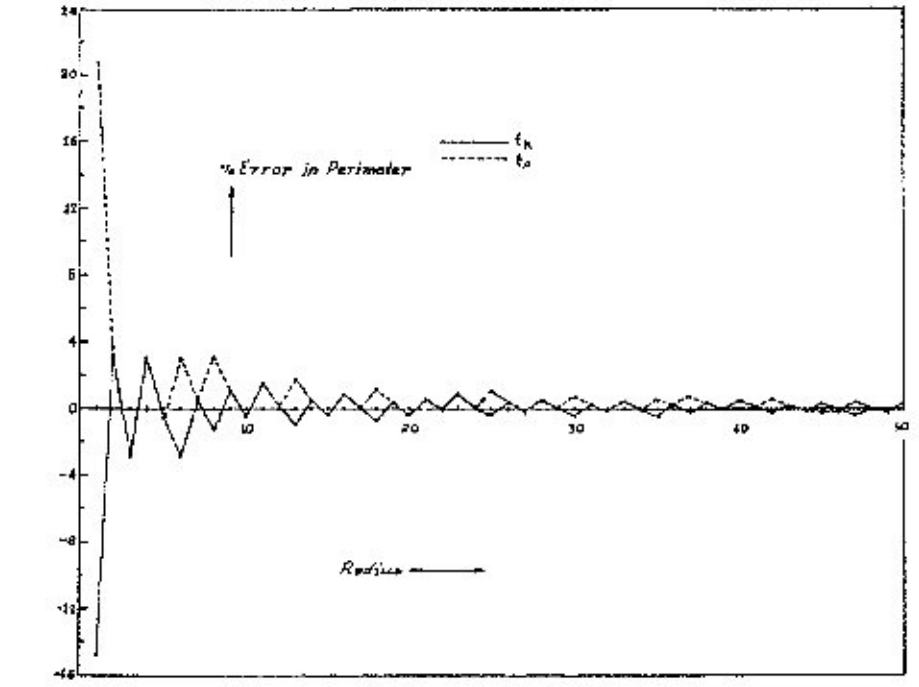
$$R(r_1, r_2, \alpha, \beta) = \bigcup_{r=r_1}^{r_2} S_r \tag{9}$$

if $r_1 = 0$ a discrete disc (d.d.) is generated. Here $S_0$ is assumed to be the center pel itself. From Proposition 2 it is easy to show that there exist no hole or gap in the d.r. or d.d. generated according to Definition 2.

Let us now compare the error of the discs generated according to Definition 2 and the disc formed by the successive generation of circles defined in [14] with the holes left unfilled. Here area is the measurement parameter and it was measured by the method given in Section 2.1. For $r > 6$, it is seen that the gaps constitute 10% of the disc area in the discs generated through method in [14], and the subsequent error is nearly 10 times larger than that in the present method. This is shown in Fig. 4.

## 3. Generation algorithm

*Circle*. For the center $(x_c, y_c)$ and radius $r$, all belonging to the set of positive integer the top-level of the algorithm for circle follows a Pascal-like language (we assume without loss of generality $min[x_c, y_c] \leqslant r$]):



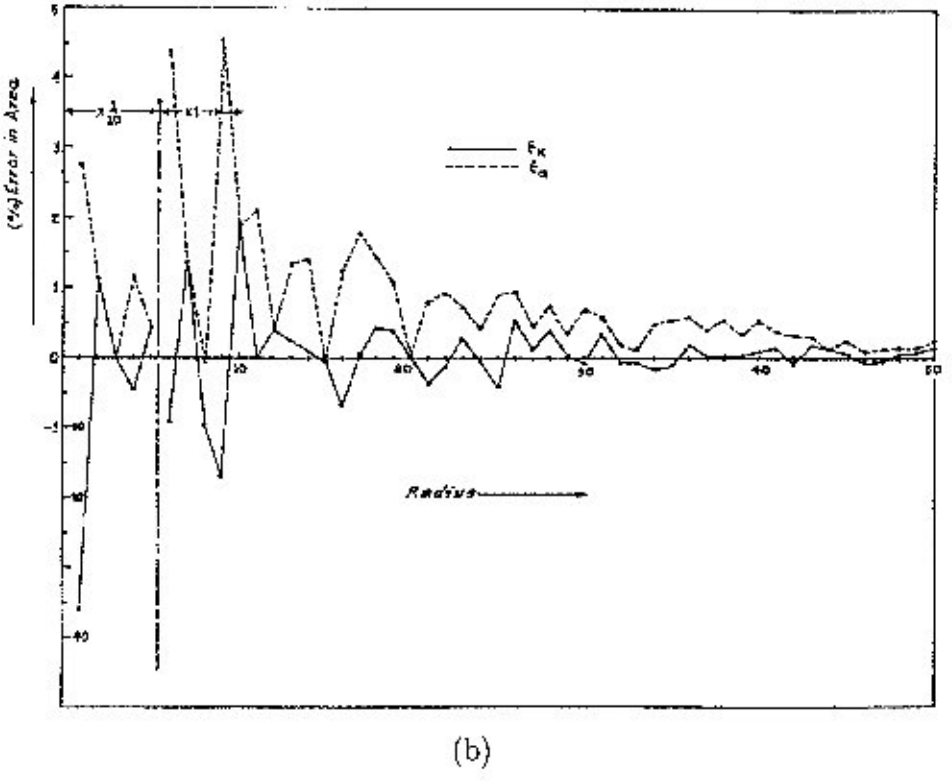




Fig. 3 . Comparison of errors (a) in perimeter (b) in area.

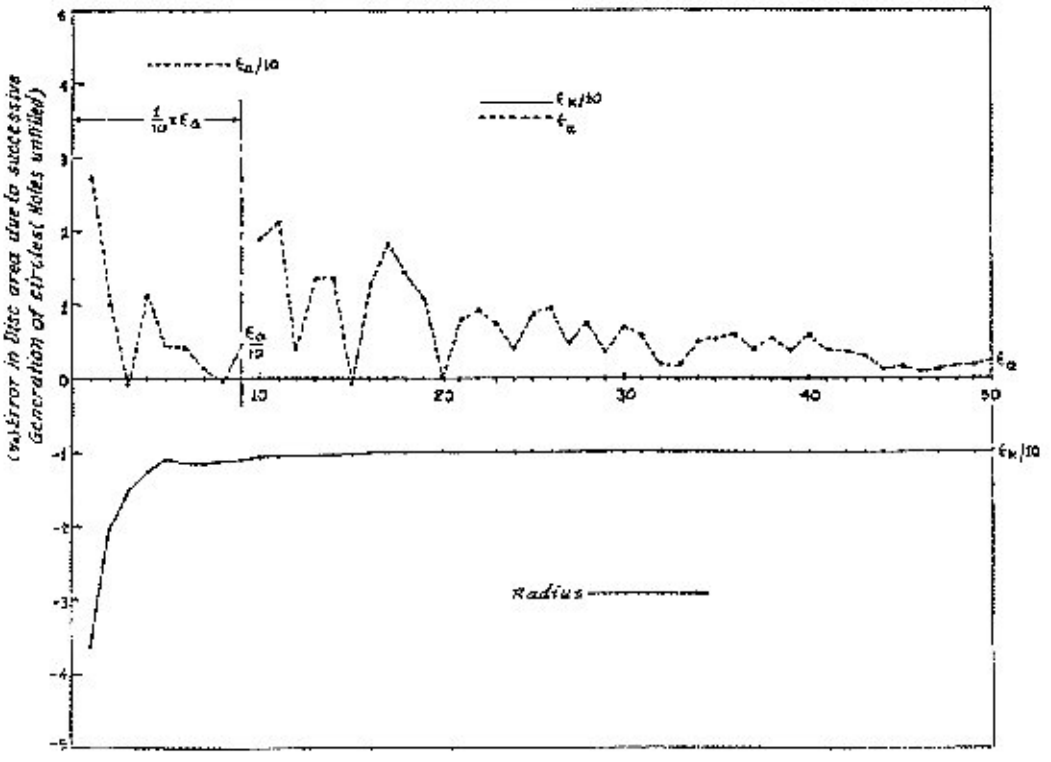



Fig. 4. Comparison of errors in disc area due to successive generation of circles.

```
var
xc, yc, r: integer;
begin
  read(xc, yc, r);
  upperfirsthalf (xc, yc, r);
  upper2ndhalf (xc, yc, r);
  lowerfirsthalf (xc, yc, r);
  lower2ndfirsthalf (xc, yc, r);
end.
```

Four different procedures plot the points belonging to the d.c. successively in the four different quadrants in the counter-clockwise direction.

```
Procedure upperfirsthalf (xc, yc, r : integer) ;
var
  x, y : integer;
  dsQ, uplimit, lolinit : real;
begin
  uplimit: = r*r + r + 0.25;
  lolimit: = r*r + r + 0.25;
        x:= xc + r;
        y:= yc;
    repeat
      dsQ: = (xc − x)*(xc − x) + (yc − y)*(yc − y);
      if (dsQ < uplimit) then
        begin
          if (dsQ > lolimit) then begin
                                    plot(x, y);
                                    y: = y + 1;
                                  end
          else
            y: = y + 1;
        end
      else
          begin
            x: = x − 1;
            y: = y − 1;
          end
    until x = xc − 1;
end.
```

The upperfirsthalf plots the points in the first quadrant. For the other quadrants the plotting algorithm is essentially the same except for some changes that take care of different conditions for $x$ and $y$.

*Ring.* For the generation of rings, the same algorithm can be used. The representation of the algorithm is given below.

```
var
  x_c, y_c, r, r_1, r_2 : integer ;
begin
    read (x_c, y_c, r_1, r_2);
    for r: = r_1 to r_2 do
      circle (x_c, y_c, r);
end.
```

The procedure for a circle is:
Procedure circle ($x_c$, $y_c$, r : integer);

```
begin
  upperfirsthalf (x_c, y_c, r);
  upper2ndhalf (x_c, y_c, r);
  lowerfirsthalf (x_c, y_c, r);
  lower2ndhalf (x_c, y_c, r);
end.
```

*Disc*. The algorithm for a disc essentially follows from the generation of a ring with $r_1 = 0$.

The generation of points in the algorithm is successive but it can be modified to generate points in the next three quadrants from the information of points in one quadrant only using the symmetry property of the d.c.

## 4. Binary contour coding

In order to code binary contours, they are, first of all, decomposed into line and convex/concave arc segments. Each contour arc, so obtained, we view as an appropriately stretched d.c. arc through affine transformation. In other words, we make a good approximation of each contour arc applying affine transformations on a discrete circular arc. To get line and arc segments from the input contour image, key pixels [5,6] are initially extracted. Key pixels on a contour are basically points of high curvature in the image plane and can be viewed as knots. The inflexion points on a contour segment between two key pixels are detected using the concept of Gaussian circle discussed in the following subsection. Any arc between two key pixels or between a key pixel and an inflexion point or vice versa is coded using the approximation information of the contour arc.

### 4.1. Detection of inflexion points

It is rather difficult to detect the points of inflection on a digital or discrete contour (a string of pixels). Due to discretization of an analog curve or contour, many inflection points may be present, although all of them may not be properly justified from the standpoint of discrete geometry in relation to discrete straight line [10–12]. We attempt to find inflexion points between two key pixels in a way somewhat similar to that in the analytic plane. This helps in maintaining the curvature of the contour during reconstruction and, as a result, the reconstruction quality is improved.

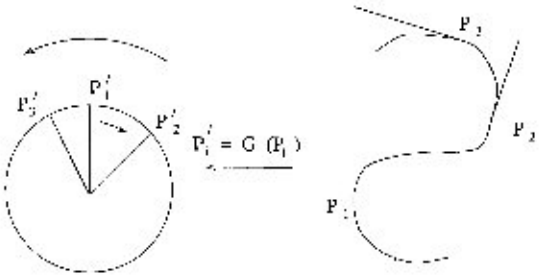



Fig. 5. Gaussian circle and its image in detecting points of inflexion.

#### 4.1.1. Gaussian circle

Consider a unit circle in the plane of a curve and draw radii in the direction of tangents at points $P_1$, $P_2$ and $P_3$, thus providing points $P'_1$, $P'_2$ and $P'_3$ as shown in Fig. 5. The process which assigns $P_i$ to $P'_i$ is known as the Gaussian map and the points on the circle is the Gaussian image of the curve. Therefore, if $G$ is the Gaussian map then

$$G(P_i) \rightarrow P'_i$$

$G$ maps every single point $P_i$ on the curve to a unique point $P'_i$ on the circle, though $G^{-1}(P'_i)$ may stand for two or more points on the curve depending on the directions of tangents at these points. Two points $P_i$ and $P_j$ appear as same under $G$ if tangents at them have same directions. In other words, it is quite likely that $G^{-1}(P'_i)$ equals $P_i$ and $P_j$ both.

Note that as we move from $P_i$ to $P_{i+1}$ and from $P_{i+1}$ to $P_{i+2}$, it is not necessary that the same sequential order is maintained by their $G$-images. With this fact, the following classification is made.

- The sequential order of the Gaussian image points $P'_i$ is the same as that of the points $P_i$ of the curve — we get regular points.
- The sequential order of $P'_i$ reverses whereas that of $P_i$'s remains the same — we get point of inflection.
- The order of $P_i$'s reverses, i.e., the direction of the tangents reverses whereas that of motion of $P'_i$s remains the same — we get Cusp of the First Kind.
- The order of $P_i$'s as well as that of $P'_i$s gets reversed — we get Cusp of the Second Kind.

Fig. 6 shows all these four classifications. In discrete domain, the tangents to the discrete curve at a point is not defined in the existing literature. Therefore, it is very difficult to get the Gaussian image of the discrete curve. To detect the approximate position of a point of inflection on a discrete contour segment between two key pixels, we first approximate the contour segment between key pixels by lines to obtain the Gaussian image. If a

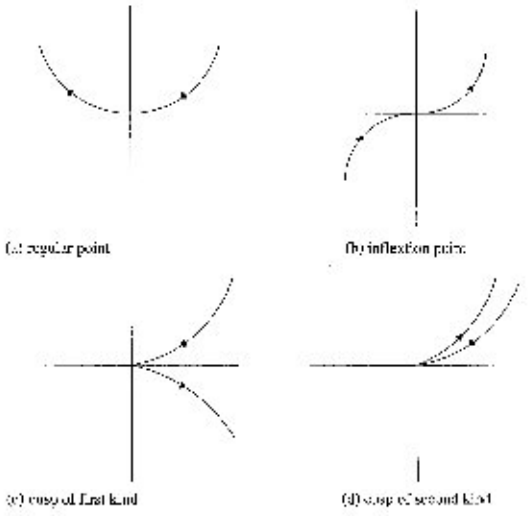


Fig. 6. Classification of different of $G$-images: (a) regular point, (b) inflexion point, (c) cusp of first kind, (d) cusp of second kind.

reversal of order in the Gaussian image is detected for any line segment then a point of inflection is marked at the midpoint of the previous line segment.

The process is repeated for all pixels between other key pixels. Thus all the key pixels and points of inflection can be extracted from the entire contour. Between any two key pixels or between a key pixel and a point of inflection or vice versa, the set of pixels can be viewed either as a line or a convex/concave arc segment. The set of pixels representing such an arc can be approximated by appropriate stretching of a d.c. arc through one or more than one affine transformations.

In order to understand stretching process, we briefly discuss affine transformation and some of their useful properties.

*4.1.2. Affine transformation*

The general form of affine transform is

$$\begin{pmatrix} x \\ y \end{pmatrix} = T\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}.$$

The coefficients $a_{11}, a_{12}, a_{21}, a_{22}, s_1$ and $s_1$ depend on translations, rotations and scalings of the object under transformation. In general,

$$a_{11} = d_1 \cos\theta, \qquad a_{12} = -d_2 \sin\phi,$$

$$a_{21} = d_1 \sin\theta, \qquad a_{22} = d_2 \cos\phi,$$

where $d_1$ is the scaling factor on $x$, $d_2$ is the scaling factor on $y$, $\theta$ is the angle of rotation on $x$ and $\phi$ is the angle of rotation on $y$. $s_1$ and $s_2$ are, respectively, the translations on $x$ and $y$.

To find the affine transformation that takes an object from one of its configurations to another configuration, we consider three points on the object in its initial configuration and their corresponding positions on the object in its final configuration. Let $(\alpha_1, \beta_1)$, $(\alpha_2, \beta_2)$ and $(\alpha_3, \beta_3)$ are the three points in the initial configuration of the object and $(\alpha'_1, \beta'_1)$, $(\alpha'_2, \beta'_2)$ and $(\alpha'_3, \beta'_3)$ are the respective three points after application of the transformation. Then, we can write the following three equations:

$$\alpha_1 a_{11} + \beta_1 a_{12} + s_1 = \alpha'_1, \tag{10}$$

$$\alpha_2 a_{11} + \beta_2 a_{12} + s_1 = \alpha'_2, \tag{11}$$

$$\alpha_3 a_{11} + \beta_3 a_{12} + s_1 = \alpha'_3. \tag{12}$$

Similarly,

$$\alpha_1 a_{21} + \beta_1 a_{22} + s_2 = \beta'_1, \tag{13}$$

$$\alpha_2 a_{21} + \beta_2 a_{22} + s_2 = \beta'_2, \tag{14}$$

$$\alpha_3 a_{21} + \beta_3 a_{22} + s_2 = \beta'_3. \tag{15}$$

*4.1.3. Some properties of affine transformation*

1. *Affine transformation preserves lines:* As the affine transformation preserves collinearity, the image of a straight line is another straight line. We, therefore, need only compute the image of the two end points of the straight line and then draw a straight line between them.

Preservation of collinearity guarantees that polygons will transform into polygons; in practice, triangles will transform into triangles. A 2D affine transformation is completely determined when its effect on a triangle is specified and any triangle can be transformed into any other by choosing the proper affine transformation. An affine transformation involves six constants and therefore has six degrees of freedom which is enough to specify how each of the three vertices of a triangle is to be mapped.

2. *Parallelism of lines is preserved:* If the two lines are parallel, their images under affine transformation are also parallel. An important consequence of their property is that parallelograms map into other parallelograms.

3. *Proportional distances are preserved:* Affine transformation has another useful property. If a point $U$ is fraction $t$ of the way between two given points $A$ and $B$ before transformation $T(\ )$ is applied, then the transformed point $T(U)$ will be the same fraction $t$ of the way between the images $T(A)$ and $T(B)$. A special case of the preservation of proportional distance is that mid points of lines, map into midpoints.

*4.2. Approximation of binary contours*

The various properties of affine transformation lends its support to successfully approximate digital contours in the image plane to any desired degree of accuracy.

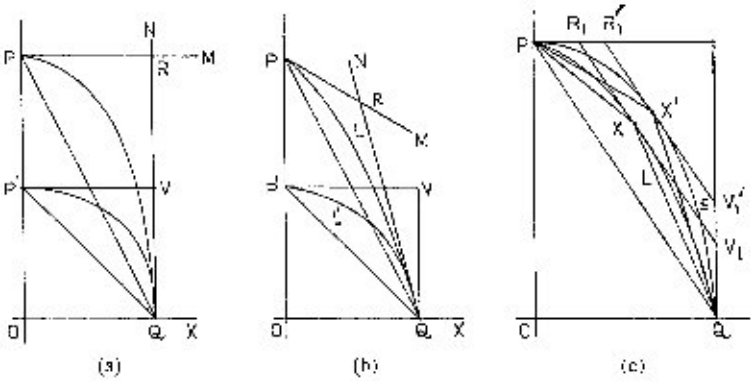



Fig. 7. First-order approximation of a typical test arc by d.c.-arc stretching with tangent configuration (a) known (b) unknown. (c) Second-order approximation of a typical contour arc.

In the first-order approximation of a contour arc, passing through two consecutive key pixels $P$ and $Q$, we consider the affine image of one quadrant discrete circular arc whose center is at the origin $O$, defined by the point of intersection of horizontal and vertical lines through the key pixels $P$ and $Q$. The axes of reference so obtained are, therefore, local to the contour arc. The lines $PM$ and $QN$ as shown in Figs. 7 (a) and (b) are slowly rotated through $P$ and $Q$ respectively towards the contour arc as long as all the pixels on the arc are on the same side of these lines. The lines $PR$ and $QR$ that the point of intersection $R$ makes with $P$ and $Q$, and the base of the arc $PQ$ defines a triangle $\triangle PQR$ that completely encloses the contour arc $PLQ$. We call this triangle the characteristic triangle of the contour arc. Lines $PR$ and $QR$ of characteristic triangle of the contour arc are the approximate tangent lines to the arc at their end points. Between $OP$ and $OQ$, either the larger or smaller line is chosen as radius of the discrete circle (d.c.) so that its one quadrant arc can be easily generated (we have chosen the smaller line $OQ$ as the radius). The one quadrant d.c. arc $P'L'Q$ enclosed within a triangle of tangent lines $P'V$ and $QV$ of the arc at its ends, and its base $P'Q$ provides a triangle for the d.c. arc. We call this triangle the characteristic triangle of one quadrant discrete circular arc. This characteristic triangle $\triangle P'VQ$ is then mapped to the characteristic triangle $\triangle PQR$ of the contour arc through the affine transformation

$$T_o: P' \to P \quad Q \to Q \quad V \to R.$$

To check the quality of approximation, we compute an error in approximation based on the difference of area bounded by the respective arc on its base. The details of error calculation is given in the next section. If the error $E_{(1)}$ in the first order approximation is greater than a desired value, say $\varepsilon$, then we seek the next higher-order approximation.

To carry out the second-order approximation, we detect the point $X'$, as shown in Fig. 7(c), on the first-order approximation arc $PSQ$, having the maximum perpendicular distance on the base $PQ$. Similarly, the point of maximum displacement on the original arc $PLQ$ is detected. Let this point be $X$. The lines passing through $X'$ and $X$ and each parallel to the common base $PQ$ intersect the approximate tangent lines to arcs at their end points, at $R_1'$, $R_1$ and $V_1'$, $V_1$ respectively. This provides two sets of triangles. In the first set we have $\triangle PX'R_1'$ and $\triangle PXR_1$ while the second set consists of $\triangle X'V_1'Q$ and $\triangle XV_1Q$. The triangle $\triangle PX'R_1'$ and $\triangle X'V_1'Q$ enclose respectively the sub-arcs $S_1^{(1)}$ and $S_2^{(1)}$ of the first-order approximation curve. Second-order approximation considers the following affine transformations.

$$T_1: R_1' \to R_1 \quad X' \to X \quad P \to P,$$

$$T_2: V_1' \to V_1 \quad X' \to X \quad Q \to Q,$$

to pull up or pull down the first-order affine arc. This provides a new affine image $T_1(S_1^{(1)}) \cup T_2(S_2^{(1)})$ which is closer to the contour arc than the first-order affine image. For the approximation of an analytic curve, this situation does not arise.

Note that the affine arc so obtained in the first- or second-order approximation, of a contour arc may not remain connected when the larger line between $OP$ and $OQ$ is taken as radius of the d.c. arc. We, therefore, need to connect them by straight line segments. This can be done using Bresenham algorithm [20]. On the other hand, if the smaller line between $OP$ and $OQ$ is chosen as radius of the d.c. arc, we may get sometimes a few excess points. These points are the interior points and should be removed (if they appear) to clean the image. Proceeding exactly in the same way as described above we can write for the approximation of a function $g(x)$ by $f(x)$, where $g(x)$ represents either a convex or concave arc on its base and $f(x)$ is the d.c. defined in one quadrant, different

following expressions for two different orders of approximation,

1st-order approximation of $g(x)$: $T^{(1)}(f(x))$;
2nd-order approximation of $g(x)$: $T^{(2)}(T_0^{(1)}(f(x)) \cup T_1^{(1)}(f(x)))$;

Carrying out of the first-order approximation is straightforward. For second-order approximation we detect the point of maximum displacement of $g(x)$ and $T^{(1)}(f(x))$ from their respective bases. Necessary affine transformations are then found out considering their characteristic triangles. Similar procedure follows hierarchically for higher-order approximation.

### 4.3. Error of approximation

In order to carry out the approximation properly, we compute an error of approximation. The error in approximation, we define as the difference in areas bounded by the original contour arc and the approximating affine arc respectively on their bases.

Let $\triangle_{p1}$ be the area bounded by a discrete circular arc in one quadrant, i.e., it is the area enclosed by intercepts $OQ$ and $OP'$ on two axes of $x$ and $y$ respectively, and the d.c. arc itself. $\triangle_{b1}$ is the area of the triangle defined by the base $P'Q$ of the d.c. arc together with $OQ$ and $OP'$ as the two other sides. This provides the area bounded by the discrete circular arc on its base $P'Q$ as

$$\triangle_o = \triangle_{p1} - \triangle_{b1} = \triangle_{p1} - \tfrac{1}{2}OQ.OP'. \quad (16)$$

Similarly, the area bounded by the contour arc between the key pixels $P$ and $Q$ on its base $PQ$ is given by

$$\triangle = \triangle_{p2} - \triangle_{b2} = \triangle_{p2} - \tfrac{1}{2}OQ.OP, \quad (17)$$

where $\triangle_{p2}$ is the area bounded by the intercepts $OP$, $OQ$ and the contour arc. It is clear geometrically that neither translation nor rotation have any effect on the area of a polygon in the analytic plane but scaling certainly does and shearing might do. In the discrete plane, however, rotation might also change the area of a polygon. As for example, a unit square when undergoes a rotation of 45°, each side of the square increases by $\sqrt{2}$, and hence the area increases by 2. When the affine transformation with the matrix $A$ is applied to an object, its area is multiplied by the magnitude of the determinant det A of $A$. Therefore,

$$\frac{\textit{Area after transformation}}{\textit{Area before transformation}} = |\det A|. \quad (18)$$

The determinant of $A$ is $|a_{11}a_{22} - a_{12}a_{21}|$. Let $\triangle_f$ the area after transformation of the discrete circular arc confined within its base $P'Q$. Since the discrete circular arc after transformation approximates the contour arc, the first-order error in approximation between two pieces of arcs can be written as,

$$E_{(1)}^2 = (\triangle - \triangle_f)^2 = (\triangle - \triangle_o|a_{11}a_{22} - a_{12}a_{21}|)^2. \quad (19)$$

Since the area of a discrete circle is not exactly the same as that of a circle in the analytic plane, we compute $\triangle$ and $\triangle_o$ with the help of Eq. (5).

### 4.4. Coding

The coding of binary contour images uses all those points which are the key and inflection points on contours. The pixels between two such points are approximated either by a line or by an arc segment. The line segment is coded in a straightforward way by coding its end point, while each arc is coded using the information of approximation. We have seen that in the characteristic triangle of contour arc, its two sides are so chosen that they are approximately tangents to the arc at end points. When these approximate tangent vectors are directed along the horizontal or vertical lines we say the tangent configuration is known and because of known tangent configuration we can easily find their point of intersection provided the end points are known. In case, they are different from horizontal or vertical lines, i.e., when the tangent configuration is unknown, the information about their point of intersection need to be stored for reconstruction of the arc. Also, on the horizontal–vertical grid, the centre of a d.c. arc corresponding to a contour arc can be easily detected. Thus, given the starting point, only one point is required to encode a straight line segment. For an arc, also one point is required when the tangent vectors have known configuration in the first-order affine approximation of a contour arc. This point is the end point of the arc. For unknown tangent configuration we need two points for reconstruction of the arc: end point of the arc and the point of intersection of the tangent lines to arc. We also require two points in the second-order approximation of the arc for known tangent configuration: end point of the arc and its point of maximum displacement measured from its base. But three points are needed to represent an arc when the tangent configuration is unknown.

For the bit allocation of points, we use the absolute co-ordinates for the starting point and relative co-ordinates for others (e.g., the end points of line and arc segments). The encoding of a segment, therefore, is based on the difference of co-ordinates of the end point $(x_e, y_e)$ from its previous point $(x_p, y_p)$ along with their respective algebraic signs. Given the previous point for a segment, we first encode the larger absolute differences between $\triangle x(= x_e - x_p)$ and $\triangle y(= y_e - y_p)$ and then the smaller one. A single bit, we take, to indicate which one greater.

For this, the largest absolute difference of $\Delta x$ and $\Delta y$ for all segments are first computed and this value is encoded by $\log_2 M$ bits (assuming the size of the image is $M \times M$), i.e., $\log_2 M$ bits convey the information of max $\{\Delta x_i, \Delta y_i\}$ where $i$ runs over all segments. Let $k = \max\{\Delta x_i, \Delta y_i\}$. Then we can always encode the larger difference of $\Delta x$ and $\Delta y$ for a segment by $\log_2 k$ bits and the smaller one by $\log_2(\Delta x + 1)$ bits or $\log_2(\Delta y + 1)$ depending on $\Delta x \geqslant \Delta y$ or $\Delta x < \Delta y$ respectively. The sign of $\Delta x$ and $\Delta y$ requires 1 bit each. For an arc, the order of approximation and configuration of tangents (known or unknown) requires 1 bit each while the point $(x_m, y_m)$ of maximum displacement from the base of arc requires $\log_2(\Delta x + 1)$ and $\log_2(\Delta y + 1)$ bits because this point can always be represented with respect to the origin of the discrete circular arc. Note that 1 has been added in the bit requirement to include the possibility of $\Delta x = 0$ or $\Delta y = 0$. Thus we get the bit requirements for an arc in the first-order approximation with known tangent configuration as

Identity of segment: 1 bit;
Oder of approximation: 1 bit;
Tangent configuration: 1 bit;
$\Delta x$ (for end point of the arc): $\log_2 k$ bits or $\log_2(\Delta y + 1)$ bits depending on ($\Delta x \geqslant \Delta y$ or $\Delta x < \Delta y$);
$\Delta y$ (for end point of the arc): $\log_2(\Delta x + 1)$ bits or $\log_2 k$ bits depending on ($\Delta x < \Delta y$ or $\Delta y \geqslant \Delta x$);
To indicate larger/smaller value: 1 bit(between $\Delta x$ and $\Delta y$);
Sign information: 2 bits (for $\Delta x$ and $\Delta y$);
Point of maximum displacement $(x_m, y_m)$:
$\log_2(\Delta x + 1) + \log_2(\Delta y + 1)$ bits;
Therefore, for first order approximation the number of bits equals for known tangent configuration: $6 + N_{bep}$ bits;
Unknown tangent configuration: $6 + N_{bep} + \log_2(\Delta x + 1) + \log_2(\Delta y + 1)$;

where

$$N_{bep} = \log_2 k + \log_2(\Delta x + 1) \quad \text{when } \Delta x \geqslant \Delta y$$
$$= \log_2(\Delta y + 1) + \log_2 k \quad \text{when } \Delta x < \Delta y.$$

For second-order approximation the number of bits
for known tangent configuration is: $6 + N_{bep} + \log_2(\Delta x + 1) + \log_2(\Delta y + 1)$,
for unknown tangent configuration is: $6 + N_{bep} + 2\log_2(\Delta x + 1) + 2\log_2(\Delta y + 1)$. If the d.c. stretching encoding scheme does not require second-order approximation for any application, then the identity bit required for approximation can be dropped out. Similar is the case for the configuration of tangents. The identity bit for this can also be dropped out if only one configuration is present throughout the approximation of all segments.

For a line segment, we require only
Identity of segment: 1 bit;
$\Delta x$ (for end point of the line): $\log_2 k$ bits or $\log_2(\Delta y + 1)$ bits depending on ($\Delta x \geqslant \Delta y$ or $\Delta x < \Delta y$);
$\Delta y$ (for end point of the line): $\log_2(\Delta x + 1)$ bits or $\log_2 k$ bits depending on ($\Delta x < \Delta y$ or $\Delta y \geqslant \Delta x$);
To indicate larger/smaller value: 1 bit;
Sign information: 2 bits;
Thus, for a line segment the number of bits required is $4 + N_{bep}$.

For the starting pixel we require $2\log_2 M$ bits and to denote the maximum of all $\Delta x$ and $\Delta y$, we require $\log_2 M$ bits. The number of bits in an application is, therefore, dynamically selected. This sometimes provides an advantage over coding techniques where fixed number of bits is always required for all segments.

## 5. Performance efficiency of the algorithm

In order to study the performance of d.c. arc stretching method in encoding binary contour images we consider the following subsection.

### 5.1. Comparison of the proposed d.c. arc stretching method

To examine the performance of coding technique by the d.c. arc stretching process a comparison has been made with the method suggested by McClure [1]. McClure suggested a technique for compressed representation of planar curves using the variable position cubic spline knots. He considered continuous, parametric representation of the curve on an interval $[a, b]$ and determined some knots on the curve for its discrete representation. In reconstructing the curve, McClure used Reinsch interpolating spline algorithm [21] which uses the information of knots and the information of weights associated with them. McClure used unity for each of the weights for stationary noise on the curve while for non-stationary noise these weights are different from unity. In order to select the knots, he [1] derived a criterion in the continuous domain. According to this criterion, knots are selected in such a way that for each $i = 1, 2, 3, \ldots, n$ the position $k(j)$ for which the function $F(k(j))$ satisfies the relation

$$\frac{i}{n} \leqslant F(k(j)) < \frac{(i+1)}{n}$$

is noted, where $n$ is the desired number of knots. McClure sets the knot point at $p_j^* = p_{k(j)}$, i.e., $x_j^* = x_{k(j)}$ and $y_j^* = y_{k(j)}$.

The signed curvature $k(t)$ in terms of parametric representation is given by

$$k(t) = |x'y'' - y'x''|/[(x')^2 + (y')^2]^{3/2}$$

and $|k(t)|^{1/3}\, \mathrm{d}s/\mathrm{d}t = |x'y'' - y'x''|^{1/3}$ at $t = j$ where $j = 1, 2, 3, \ldots, N$. The function $F$ is given by

$$F(\tau) = \int_a^{\tau} |k(t)|^{1/3} \frac{\mathrm{d}s}{\mathrm{d}t}/L.$$

$F(\tau)$ is basically the normalized affine-arc length between two positions because

$$L = \int_a^b |k(t)|^{1/3} \frac{\mathrm{d}s}{\mathrm{d}t}$$

in the above expression is the total affine arc length of the curve. In the discrete domain, replacing the integral sign by the summation sign, we get

$$L = \sum_{j=1}^{N-1} |\triangle x_j \triangle^2 y_j - \triangle y_j \triangle^2 x_j|^{1/3}$$

and

$$F(k(j)) = \sum_{j=1}^{N-1} |\triangle x_j \triangle^2 y_j - \triangle y_j \triangle^2 x_j|^{1/3}/L.$$

McClure [1] defined an empirical distribution $F_n(t)$ of the discrete parameter values associated with the points $\{p_i^*\}_{i=1}^n$, i.e., for each $t_i$ there corresponds a point $p_i^* = (x(i), y(i))$. This distribution, he wrote as

$$F_n(t) = i/n \quad \#(t_i : t_i \leqslant t)$$

and he proved $\lim_{n\to\infty} F_n(t) = F(\tau)$. The asymptotic equivalence of $F_n$ and $F$ means that the $i$th point $p_i^*$ of the $n$-point discretization will be identified with a parametrization $t_i^*$ at which $F \approx i/n$. Since

$$|k(t)|^{1/3} \frac{\mathrm{d}s}{\mathrm{d}t}$$

is strictly positive $F$ has a unique inverse $F^{-1}$. Thus $t_i^* = F^{-1}(i/n)$. This parametrization is helpful in Reinsch interpolating spline algorithm because $t_i^*$ with $i = 1, 2, 3, \ldots, n$ satisfies

$$t_1^* < t_2^* < t_3^* < t_4^* < \cdots < t_n^*.$$

The number of knots $n$ was selected considering

$$\sum_{i=1}^{N} (x_i - x_r(i))^2 w_i \leqslant s_x,$$

$$\sum_{i=1}^{N} (y_i - y_r(i))^2 w_i \leqslant s_y,$$

where $(x_i, y_i)$ and $(x_r(i), y_r(i))$ are respectively the observed data points and the generated points by Reinsch [21] algorithm. McClure considered $s_x = s_y = s$ and $\{w_i\}_{i=1}^n = 1$ for stationary noise.

It is, therefore, clear that in general one needs $2n$ points or $\alpha$ bits for a $M \times M$ contour (binary) image with stationary noise and $\alpha + \beta$ bits for a non-stationary binary contour image, where $\alpha$ and $\beta$ denote the number of bits for the co-ordinates of a point and its weight respectively. Due to quantization, different errors may be incurred due to different spline parameter values. Post processing or cleaning of the image may be required due to small parameter values.

## 6. Results and discussion

Figs. 8(a), 9(a) and 10(a) are three different figures. They are the contours of a butterfly, chromosome and numeral "8". These contour images are taken as inputs to our coding algorithm. The key pixels and the points of inflexion, first of all, are extracted from these contours and they are marked by 3 and I respectively.

Fig. 8(b) is the butterfly image regenerated by the d.c. stretching method while Fig. 9(b) is the regenerated

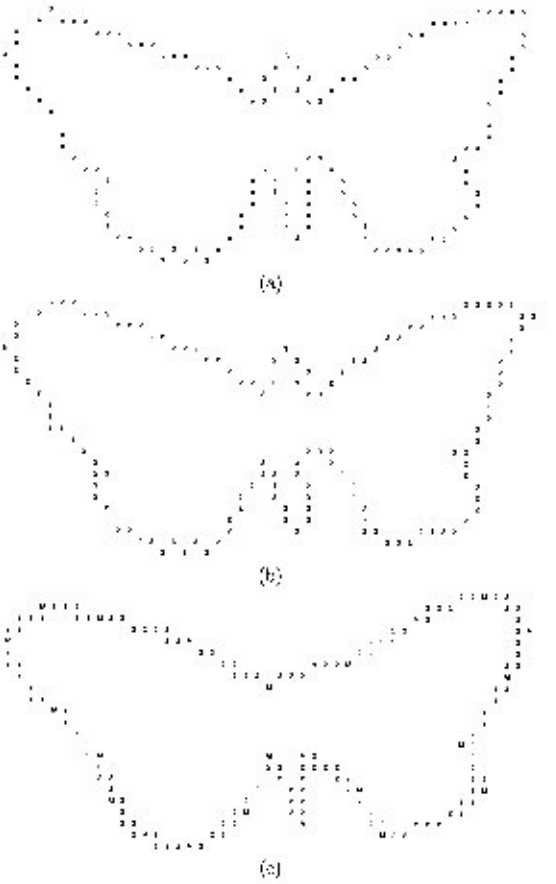


Fig. 8. Butterfly image: (a) input, (b) output by d.c. stretching, (c) output by McClure's method.

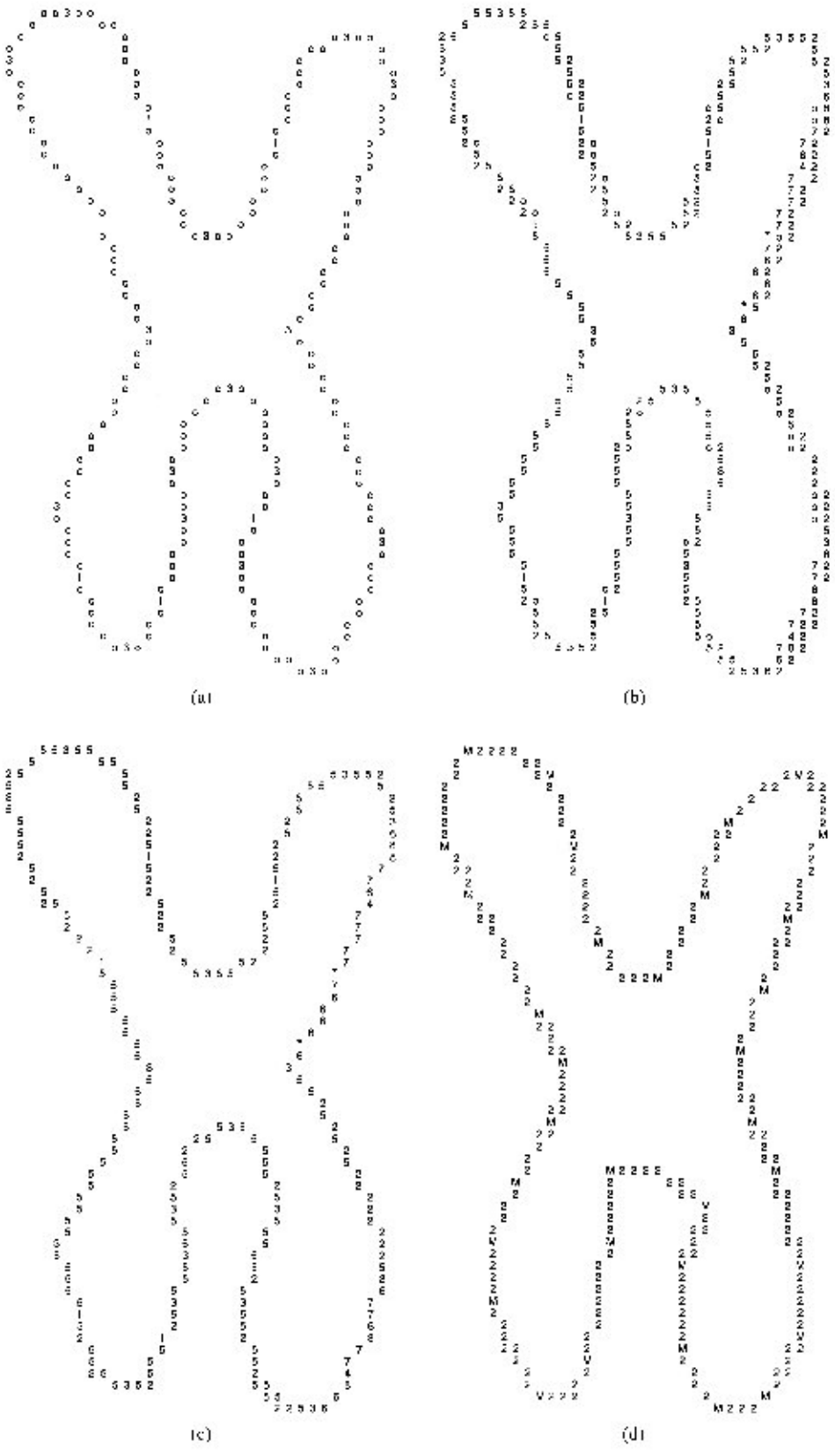



Fig. 9. (a) Input chromosome image. (b) Regenerated chromosome image by d.c. stretching method (before cleaning). (c) Regenerated chromosome image by d.c. stretching method (after cleaning). (d) Regenerated chromosome image by McClure's method.

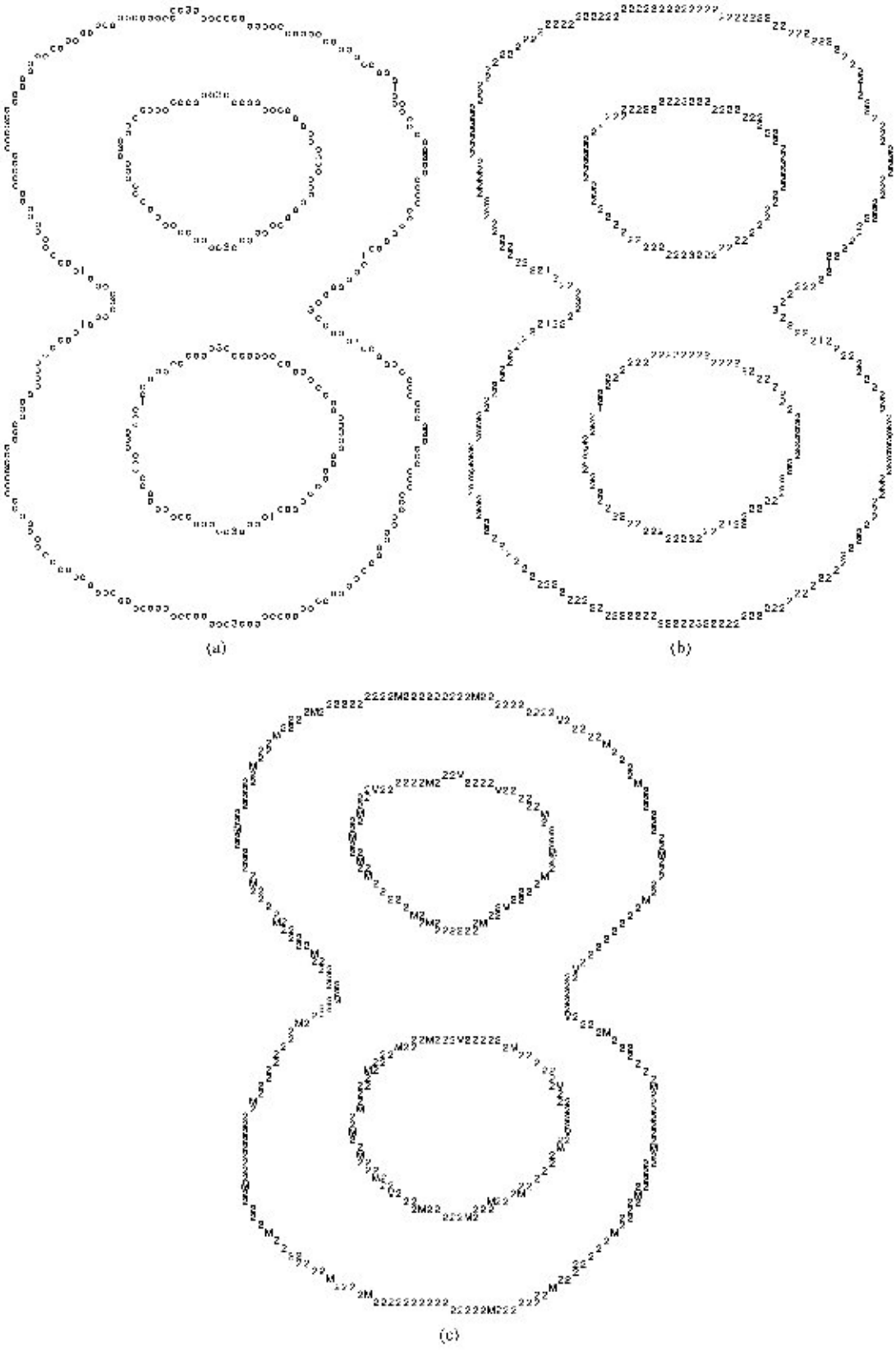


Fig. 10. (a) Input numeral "8" image. (b) Regenerated numeral "8" image by d.c. stretching. (c) Regenerated numeral "8" image by McClure's method.

Table 1
Bit requirements

| Figure | Bit requirement in d.c.-stretching | Bit requirement McClure's method | Compression efficiency $\delta_{relative}$ (%) |
|---|---|---|---|
| Butterfly | 406 | 288 | 70.93 |
| Chromosome | 346 | 396 | 114.45 |
| Numeral "8" | 398 | 806 | 202.51 |

Table 2
Computation of error in area and compactness

| Input contour image | Percentage error in area | | Percentage error in compactness | |
|---|---|---|---|---|
| | d.c. Stretching | McClure | d.c. Stretching | McClure |
| Butterfly | 0.4219 | 1.2658 | 1.7058 | 29.4685 |
| Chromosome | 5.0144 | 6.7435 | 2.1449 | 17.1352 |
| Numeral "8" | 0.1217 | 0.1217 | 3.6648 | 4.9186 |

version of the chromosome image. To examine the proximity between the input and output, we have superimposed the regenerated image on its input. In order to show this proximity, the letters o, 2, 3, I, 4, 5, 6, 7 and $*$ have been used to represent both the input and output images. o, 3, I are the pixels in the input image; 3 and I being the key pixels and pixels representing points of inflexion. Since, approximation is always carried out between two key pixels, or between a key pixel and a pixel of inflexion or vice-versa, 3 and I are always present in both the input and output images. 2 and $*$ are the pixels generated by the first- and second-order approximations respectively. Hence, these pixels are basically the output pixels. 5 indicates pixel positions where the pixels denoted by 2 coincide with o. 6 denotes pixel position where the input pixel is identical to its first-order approximation pixel 2 while 7 denotes the pixel position where the input pixel is identical to its second-order approximation pixel $*$. Letter 4 is the pixel on input contour segment where the corresponding first-order approximation pixel with maximum displacement is mapped during its second-order approximation. Thus, 4 subdivides the first-order approximated segment into two other segments for second-order approximation. Note that some of the pixels in the output image generated through approximations are interior pixels and need to be cleaned. Fig. 9(c) is the clean version of Fig. 9(b). The regenerated images for other two images are shown in Figs. 8(b) and 10(b), respectively. Figs. 8(c), 9(d) and 10(c) are the images obtained through McClure's method [1].

The number of bits for different contour images has been computed in each case and is shown in Table 1. The Table 1 provides a comparison between McClure's method [1] and ours. Also to examine the quality of regenerated binary images we have computed the area difference and compactness of both the input and output images. Table 2 shows the percentage error in area and compactness.

The relative comparison shows that the proposed method is either superior or is comparable to McClure's method. For the butterfly image, the number of bits was found to be less in McClure's method and the error in area is also comparable to that of our method but the compactness value shows a high percentage of error which indicates the quality of the regenerated butterfly image is seriously affected. This is prominent from Fig. 9(c). Fig. 9(d) also shows the quality of the chromosome image regenerated by d.c. stretching method is better than that of McClure's method, while for numeral "8" both the methods produce almost same quality (quality is slightly better by the d.c. stretching method) the developed method requires half the number of bits compared to that required by McClure's method. Also, it is seen that cleaning is required in all images regenerated by McClure's method.

In order to examine how far the proposed approximate coding technique is away from the exact one, we have also compared the results of the d.c. stretching method with that of CRLC [2], DLSC [3], and Eden and Kocher's [4] methods. CRLC scheme views an input contour as a chain of links and compresses the number of such links while DLSC scheme considers straight line segments of a fixed class to represent an input contour. Eden and Kocher's method considers a 4-connected contour for an input and it requires 1.2 bits per node.

Table 3
Bit requirements

| Figures | Number of bits | | | | Compression efficiency | | |
|---|---|---|---|---|---|---|---|
| | CRLC | DLSC | Eden | d.c.-Stretch | $\delta_{CRLC}$ | $\delta_{DLSC}$ | $\delta_{Eden}$ |
| Butterfly | 799 | 531 | 247 | 406 | 196.79 | 130.78 | 60.83 |
| Chromosome | 1175 | 603 | 355 | 346 | 339.59 | 174.27 | 102.60 |
| Numeral "8" | 2085 | 1169 | 629 | 398 | 523.86 | 293.7 | 158.04 |

Table 3 sows a comparison of bit requirements for the three different Figs. 8(a), 9(a) and 10(a) and their relative compression efficiency computed by the proposed method relative to CRLC, DLSC and Eden and Kocher's methods. It shows a good amount of storage efficiency provided one accepts the quality compromise.

## References


[1] D.E. McClure, Copmutation of approximately optimal compressed representations of discrete plane curves, in: Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing, 1977, pp. 175–182.

[2] T.H. Morrin, Chain link compression of arbitrary black and white images, Comput. Graphics Image Process. 5 (1976) 172–189.

[3] M.K. Kundu, B.B. Chaudhuri, D. Dutta Majumder, A generalized digital contour coding scheme, Comput. Vision Graphics and Image Process. 30 (1985) 269–278.

[4] M. Eden, M. Kocher, On the performance of a contour coding algorithm in the context of image coding part I: contour segment coding, Signal Process. 8 (1985) 381–386.

[5] S.N. Biswas, S.K. Pal, D.D. Majumder, Binary contour coding using Bezier approximation, Pattern Recognition Lett. 8 (1988) 237–249.

[6] S. Biswas, S.K. Pal, Approximate coding of digital contours, IEEE Trans. System, Man Cybernet. 18 (1988) 1056–1066.

[7] F. Ghorbel, M. Daoudi, A. Mokadem, O. Avaro, H. Sanson, Global planar rigid motion estimation applied to object-oriented coding, Proceedings of the ICPR, Vol. 2, 1996, 641–645.

[8] F. Ghorbel, V. Burdin, Invariant approximation of star shaped form for medical application, in: P.J. Laurent, A.L. Mehaute, L.L. Schumaker (Eds.), Curves and Surfaces II, Academic Press, New York, 1994.

[9] H. Freeman, Boundary encoding and processing, in: B.S. Lipkin, A. Rosenfeld (Eds.), Picture Processing and Psychopictories, Academic Press, New York, 1970.

[10] A. Rosenfeld, Digital straight line segment, IEEE Trans. Comput. 23 (1974) 1264–1269.

[11] L.D. Wu, On the chain code of a line, IEEE Trans. Pattern Anal. Mach. Intell. 3 (1982) 347–353.

[12] R. Brons, Linguistic methods for description of a straight line on a grid, Comput. Graphics Image Process. 2 (1974) 48–62.

[13] L. Hodes, Discrete approximation of continuous convex blobs, SIAM J. Appl. Math. 19 (1970) 477–485.

[14] M. Doros, Algorithms for generation of discrete circle rings and disks, Comput. Graphics Image Process. 10 (1979) 366–371.

[15] Z. Kulpa, M. Doros, Freeman digitization of integer circles minimizes the radial error, Comput. Graphics Image Process. 17 (1981) 181–184.

[16] Z. Kulpa, On the properties of discrete circle rings and disks, Comput. Graphics Image Process. 10 (1979) 348–365.

[17] B.W. Jordan, W.J. Lennon, B.D. Holm, An improved algorithm for the generation of nonparametric curves, IEEE Trans. Comput. 22 (1973) 1052–1062.

[18] S.N. Biswas, B.B. Chaudhuri, On the generation of discrete circular objects and their properties, Comput. Vision, Graphics Image Process. 32 (1985) 158–170.

[19] Z. Kulpa, Area and perimeter measurement of blobs in discrete binary pictures, Comput. Graphics Image Process. 6 (1977) 434–451.

[20] J.E. Bresenham, Algorithm for computer control of a digital plotter, IBM System J. 4 (1965) 25–30.

[21] C.H. Reinsch, Smoothing by spline functions, Numer. Math. 10 (1967) 177–183.


**About the Author**—SAMBHUNATH BISWAS obtained the M.Sc. degree in Physics from University of Calcutta in 1973. He was in electrical industries in the beginning as a Graduate Engineering Trainee and then as a Design and Development Engineer. He was a UNDP fellow at MIT, USA, to study Machine Vision in 1988. He visited the Australian National University at Canberra in 1995. He is now a programmer at the Machine Intelligence Unit in Indian Statistical Institute, Calcutta. His research interests include image processing, machine vision, computer graphics, and pattern recognition.