Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Abstract

A Heuristical Method Of Corner Detection On an Image Boundary

By

Qiulin Li

A heuristics-based method of corner points detection on an image boundary is proposed and implemented. The method uses the sampled boundary distances to find all of the candidate corner points along the image boundary. Then the curvature characteristical value is used to measure the severity of curvature change of the candidate points. Those candidates whose curvature characteristical value is under some threshold are eliminated. The paper also proposed some mechanism to reduce the effect of noises on the boundary. Experiments show that it is an efficient method and it gives satisfactory results on some image boundaries.

A Heuristical Method Of Corner Points Detection On An Image Boundary

By

Qiulin Li

A Thesis
Submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information

Jan. 1992

Approval Page

A Heuristical Mathod Of Corner Points Detection On An Image Boundary

by

Qiulin Li

Dr. David T. Wang, Thesis Advisor
Assistant Professor of Computer and Information Science Department,
New Jersey Institute of Technology

Biographical Sketch

Author: Qiulin Li

Degree: Master of Science in Computer Science

Date: Jan. 1992

Undergraduate and Graduate Education:

1. Master of Science in Computer Science, New Jersey Institute of Technology, Jan. 1992.
2. Bachelor of Science in Computer Science, Hefei University of Technology, PR China, Jan. 1982.

Major: Computer Science

Acknowledgement

The author would like to thank Dr. David T. Wang who advised this research work. Without his instructions it is impossible that this work is done. The author would also like to thank the visiting professor Guizhang Tu of the Computer and Information Department of New Jersey Institute of Technology who helped the author in solving the mathematical problems of this research work.

Table of Contents

1. Introduction

The recognition of geometrically shaped objects by their boundaries or contours is of great importance in the field of computer vision and pattern recognition. For example, if a person is shown a silhouette formed by back illuminating a pile of objects, it is usually a simple task to identify the objects that contribute to the boundary. Even though there are no depth or color cues available, we can eventually recognize the objects from their two dimensional boundary contours[1].

To analyze and recognize a two dimensional image boundary, it is important to identify those basic geometrical elements which contribute to the combination of the image boundary. Asada and Brady[1] categorized those basic elements into five types, and called them *curvature primal sketch*. They are: *corners, smooth joins,*

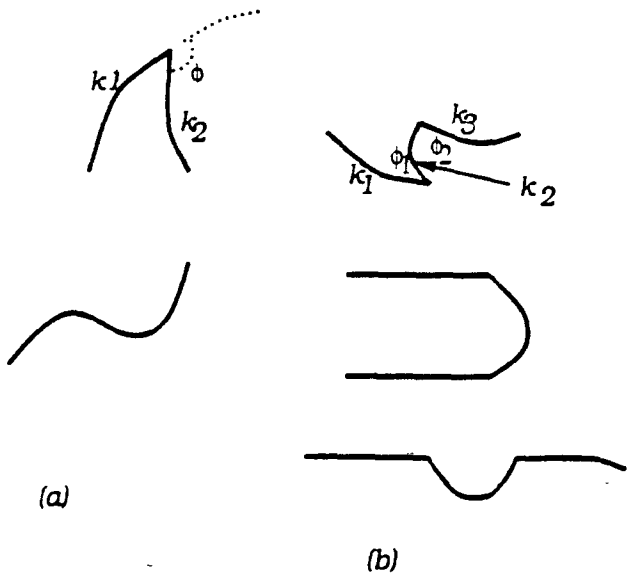


Fig.1. The curvature changes proposed as primitives are illustrated. (a) The corner and smooth join are isolated changes. (b) The crank, end, and bump or dent are compound changes formed of corners. (After Asada and Brady[1])

cranks, ends, and bumps or dents. Fig. 1 illustrates these five basic elements of boundary changing. The elements represent five different forms of curvature change

along the boundary of the object. For example, corner represents a discontinuity point of the curvature change. It isolates two boundary segments, each of which has a continuous change of curvature. Smooth join, on the other hand, isolates also two segments of boundary with continuous curvature change, except that the curvature at this point is "smooth". If we describe this fact in mathematical language, it is just to mean that the first order derivative of the boundary function is continuous at the smooth join but discontinuous at the corner. For another example, both cranks and ends have two close discontinuity points of curvature changes. The difference is that the former has the change in opposite directions while the latter has the changes in the same direction.[1]

Study the five basic elements, we see that each of them describes a special case of severe curvature change along the image boundary. The most important point here is the severity of the curvature changes. From the elements we see that each element contains at least one such point. We could call this kind of points *corner points* (We call this corner points to differentiate it from the corners in the previous paragraph).

The presence of a corner point is usually an indication of feature point in the image boundary, a road intersection, a house, a cultivated field. etc. If a surface region is broken into many parts and then reassembled, at least one of them (and possibly all) will exhibit a "corner" at every point at which n parts join to form a closed junction.

From the above discussion we can get such a conclusion: Corner points are the important feature points on the image boundaries. Therefore if we could recognize all of the corner points along the image boundaries, we can then gain a lot of important information for the image analysis and classification.

In this paper we present a heuristics based method for the corner points detection. It uses the sampled boundary distances and the ratio of the height to the width of a piece of curve at the neighborhood of a possible high-curvature point. The experiments show that it is an efficient and promissing method.

The paper is arranged in the following way: Chapter 2 reviews some of the previous research work on the topic in that seven methods are discussed and evaluated. Chapter 3 introduces our heuristical corner points detection algorithm. Chapter 4 discusses some issures of implementation of the method. Chapter 5 shows some experimental resulte and some conclussions.

2. Survey of Previous Work on Corner Points Detection

While the definition of the corner points is very simple, it is not a trivial work to detect them along the image boundaries. In the past two decades, a lot of research work has been spent in this field. We survey some of it.

There are two directions in this field. One refers to those which are image-based, the other refers to those which are boundary-based. In this paper, we pay attention to the latter.

Liu and Srinath[2] summarized and evaluated 6 methods of corner points detection based on the image boundaried represented in chain-code. Following are brief description of these methods:

Method 1: *Medioni-Yasumoto Corner Detector.*

In this method, an image boundary is fit by a parametric cubic B-spline[5] and use the displacement between the original point and the interpolating spline to decide whether the point is a corner point. Those points for which the displacement exceeds a given threshold and the curvature is high are recognized as corner points.

The experiments showed that this is very sensitive to the smoothness of the boundary points. This is because the scheme detects the corner points by using five points in the boundary to compute the curvature and displacement of its B-spline fit. Thus, if any of these 5 points moves by even one pixel, the result can change drastically.

Method 2: *Beus-Tiu corner detector.*

In this method, a corner point is defined as an isolated discontinuity (local curvature) in the mean slop whose prominance is proportional to the length of the discontinuity-free regions to either side as well as the severity of discontinuity. If we define δ_j^s as the measure of servrity of discontinuity, whose computation was described in [2] and [7], and t_1 and t_2 are the length of the discontinuity-free region in both sides. Then we can calculate the value of decision function K_j as follows:

$$K_j = f(t_1) \times \sum_{i=j}^{j+s} \delta_i^2 \times f(t_2),$$

where f is some function like *square root*, ln, etc. The decision rule is as follows: If K_j is above a given threshold, then j is considered as a corner point.

As shown in Beus and Tiu, this algorithm fails to detect some obvious corner points and detects spurious corner points in some situation. Some improvements are also proposed to enhance this method.

Method 3: *Weighted-K-curvature corner detector.*

Rutkowski and Rosenfeld concluded that the result of this method can detect corner points most closely resembling those detected by a human. In this method, the contour is defined by a chain code. Given a set of weights, $w_1, w_2, ..., w_k$, the weighted-K-vector at point i is defined as:

$$\mathbf{V}_i^{(1)} = \sum_{j=1}^{k} w_j \mathbf{v}_{i-j},$$

$$\mathbf{V}_i^{(2)} = \sum_{j=1}^{k} w_j \mathbf{v}_{i+j-1}.$$

Then the weighted-K-curvature at this point could be determined as:

$$\theta_i = \cos^{-1}\left\{ \frac{\mathbf{V}_i^{(1)} \cdot \mathbf{V}_i^{(2)}}{|\mathbf{V}_i^{(1)}|\,|\mathbf{V}_i^{(2)}|} \right\},$$

where $\cdot$ denotes the vector inner product and $||$ denotes the norm. Once the curvature has been estimated for all contour points, a corner point is detected if the curvature is above a given threshold and is a local maximum within a range $[i-k, i+k]$.

The result showed that only very few spurious corner points are detected, but it also fail to detect some real corner points.

Method 4: *Rosenfeld-Johnston Corner Detector[8]*

In this method, a so-called *K-cosine* value at the point $i=(x_i, y_i)$ is calculated in this way:

$$c_{ik} = \frac{(\mathbf{a}_{ik} \cdot \mathbf{b}_{ik})}{|\mathbf{a}_{ik}|\,|\mathbf{b}_{ik}|},$$

where $\mathbf{a}_{ik}, \mathbf{b}_{ik}$ are called *K-vectors* in the point i, which is defined as:

$$\mathbf{a}_{ik} = (x_i - x_{i+k}, y_i - y_{i+k})$$

$$\mathbf{b}_{ik} = (x_i - x_{i-k}, y_i - y_{i-k}).$$

They also defined a way to choose an appropriate value of k. Assume the maximal value is m. Then using the *K-cosine* definition to compute the values $c_{i1}, c_{i2}, ..., c_{im}$, the best value of k is chosen such that the following holds:

$$c_{im} < c_{i,m-1} < ... < c_{ik} \geq c_{i,k-1}.$$

The value of k and the corresponding c_{ik} are used to detect corner points. If for all j such that $|i-j| \leq \frac{k}{2}$, c_{ik} is a local maximum or minimum, point i is considered to be

a corner point.

The drawback of this method is that it can lead to incorrect detection results when corner points occur too close to one another.

Method 5: *Rosenfeld-Weszka corner detector.*

This is just the modification of method 4. It smoothes the *K−cosine* at each point by a process of averaging. The *average* of *K−cosine* is defined by:

$$\bar{c}_i^{(k)} = \frac{1}{k+2}[c_i^{(k)} + c_i^{(k-1)} + \ldots + c_i^{(k/2)}]$$

for k even, and

$$\bar{c}_i^{(k)} = \frac{1}{k+3}[c_i^{(k)} + c_i^{(k-1)} + \ldots + c_i^{((k-1)/2)}]$$

for k odd. The $\bar{c}_i^{(k)}$ is then used as a decision function in place of c_{ik} of the previous method.

Method 6: *Cheng-Hsu corner detector*

This method is based on the definition of so-called bending degree according to the direction changes of forward and backward arms. Because it needs more paragraphs to describe the method, we omit the description. The interested reader could reference [2] and [ ].

The experiment results showed some drawbacks of the method. First, it is also, as of method 1, very sensitive to the smoothness of the boundary. On a noisy image, this can lead to spurious corner points. Second, it may result in a multiple response to the same corner point. Very often there is no way to find local maximum and to localize the corner point.

Apart from the methods described above, there are also some other methods for the points detection. One is to use the difference of slope (DOS)[3],[4]. to estimate the curvature of the boundary points.

In this method, curvature at a point is estimated as the angular difference between the slope of the line segments fit to the data before and after the point. For the DOS method, both segments are of arc length W, and the overlap of the two segments is denoted as M. The total length from the beginning of one segment to the end of another segment could be written as:

$$L = 2W + M.$$

Thus, if $M > 0$, there is an overlap between the segments. If $M = 0$, it means that the

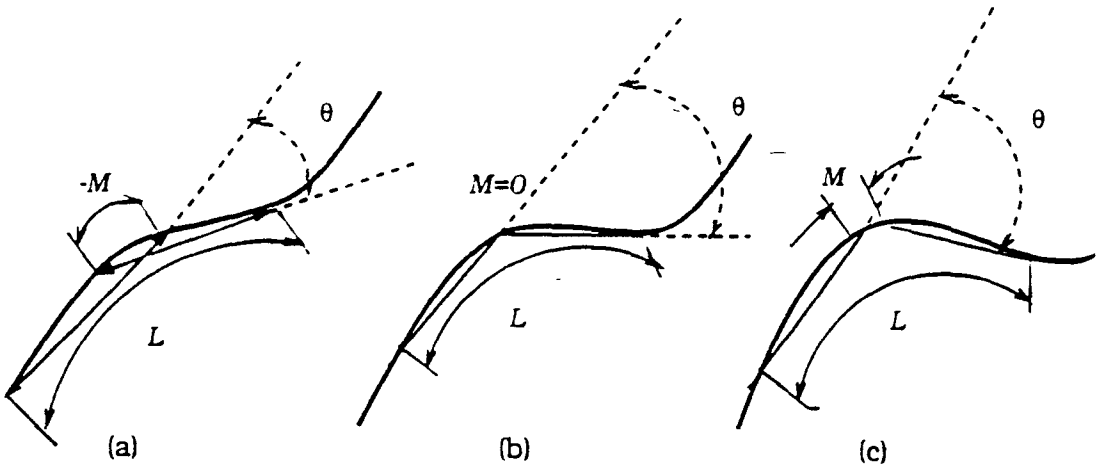


Fig.2. DOS method for a) M<0, b) M=0, c) M>0. (After O'Gorman[4])

two segments are joined together. And if $M < 0$, the two segments are separate apart. Fig. 2 shows the three cases of M. We usually denote the DOS method with $M > 0$ as DOS^+ method. For the DOS^+ method, first the orientation of the two line segments connecting the endpoints of the arcs are calculated. The formula for the orientation is:

$$\gamma_m = \tan^{-1}\left(\frac{\Delta y_m}{\Delta x_m}\right) + \begin{cases} 0, \text{ for } \Delta y_m \geq 0,\ \Delta x_m \geq 0 \\ \pi, \text{ for } \Delta y_m \geq 0,\ \Delta x_m < 0 \\ 0, \text{ for } \Delta y_m < 0,\ \Delta x_m \geq 0 \\ \pi, \text{ for } \Delta y_m < 0,\ \Delta x_m < 0 \end{cases} \text{for } m=1, 2,$$

where $\Delta y_m = y_{mf} - y_{mo}$, $\Delta x_m = x_{mf} - x_{mo}$, and (x_{mo}, y_{mo}), (x_{mf}, y_{mf}) are the starting and ending points of the line segments respectively.

After the orientation are calculated, the difference of the slopes is calculated according to the formula:

$$\theta_i = \gamma_2 - \gamma_1.$$

The geometrical meaning of θ_i is that it is the supplement of the corner angle. This means that the smaller the corner angle, the larger the corresponding value of θ_i. We can determine by using the value of θ–plot whether a boundary segment is a straight line, a corner, or a curve. For a boundary segment to be treated as a straight line, the

value of θ–plot should not exceed the range from $-\theta_{s_{max}}$ to $\theta_{s_{max}}$, where $\theta_{s_{max}}$ is usually defined as:

$$\theta_{s_{max}} = \tan^{-1}\frac{2}{L-M-2\sqrt{2}+2}$$

If the θ–plot value in a boundary exceeds this range, then it is considered that the segment is a curve or contains a corner. In this case, a peak width is determined. To differentiate the corners from the curves, some task-dependent knowledge is usually used. For example, often only corners of 90° and curves of much lower curvature are present. Thus a simple threshold on the θ–plot peaks is sufficient to distinguish corners from curve features. If the feature segment is decided to contain a corner, the coordinates of the corner point could then be calculated. If it is decided to contain a curve, the curvature center could also be calculated. The readers may look for [3] for more details of the calculations.

In [4] some comparisons of the *DOS*$^+$ method with the Gaussian smoothing method has been presented. The results showed that the *DOS*$^+$ method has better corner detectability over the Gaussian smoothing method. The other advantage of *DOS*$^+$ is that it performs better than Gaussian smoothing method under the noise distortion of the boundary.

From the above brief survey we could see that the fundalmental characteristics of a corner point is its change of curvature along the boundary. According to the calculus, the curvature of a point in a curve could be defined as:

$$\frac{d^2y/dx^2}{(1+(dy/dx)^2)^{3/2}}$$

But in general we could not apply this formula directly in the computations. There are at least two reasons for this. First, the above formula is derived from the continuous mathematics, while the boundary of the image is usually consisted with finite number of discrete points. Until now we could still not predict what would happen if we apply this continuous formula to the discrete points. Second, our boundaries are usually subjected with distortion of noises. It may cause great change for the behavior of the formula even though a very tiny noise is present.

We could also see that almost all of the methods described above are based to some extend on some heuristics. Because of this, every method has some advantages as well as some drawbacks. They usually succeeded in one set of boundaries while failed in another. From this point of view we can say that more research work is still

needed in development more efficient and more precise methods for the corner points detection.

3. The Heuristical Algorithm for Corner Points Detection

3.1. General Discussion

Consider a piece of boundary segment in an image whose boundary points are represented in Cartesian coordinates. We connect the end points of the segment with a straight line. Then we consider the distances of the boundary points to this straight line segment (We usually call these distances the sampled boundary distances according to [12]). By studying these distances we could find that those points whose distance is a local maximum or local minimum in a neighbor usually represent a meaningful change of the curvature along the boundary. Fig. 3 described this idea in an intuitive way. In Fig. 3, the boundary points a, d, and e may be considered as feature points. The most obvious characteristics is that the Sampled Boundary Distances (SBD for short) for those points, that is, the distances from these points to the straight line, l_1, l_4, l_5 are either a maximum (l_1, l_5), or a minimum (l_4) in a neighbor area around these points. This suggest us that we can use this characteristics to help

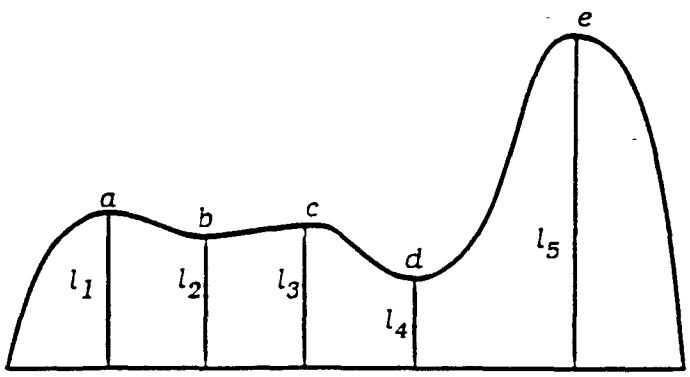


Fig. 3. Boundary points with local maximal or local minimal SBD's may be considered as candidates of feature points.

us to determine the candidates of corner points on an image boundary.

Only this condition is not enough to consider these candidates as corner points. Let take as an example the boundary points d and e in Fig. 3, whose SBD's are also local maximum or local minimum in their neighbor areas. Because the curvatures on these points are so "smooth", we usually don't think that they more valuable feature than other boundary points. Therefore we have to eliminate them. This could be done by further considering the curvature characteristics of the candidate points. According to the differential geometry, we could define this curvature characteristics as a ratio which is formulated as follows:

$$k_i = \frac{y_i}{x_i^2}, \tag{1}$$

where y_i is the height of a piece of boundary curve in the neighbor of the point p_i, and x_i is the width of that segment. We will give the detailed development of the decision formula in the next subsection.

3.2. The Parabolic Property in the Neighborhood of a Point on a Curve

From the theory of differential geometry, we realize that the following statement holds:

Assume that $\vec{r} = \vec{r}(s) = (x(s), (y(s))$ is a curve on plane XOY, where s is the arc length. Without loss of generality, we can assume that the origin is located at $\vec{r}(0)$, i.e., $x(0) = y(0)$. If the curvature of curve $\vec{r}$ at $s = 0$ is $k > 0$, the parametric equation of the curve in the neighborhood of this point is:

$$\begin{cases} x(s) = s + o(s^2), \\ y(s) = \dfrac{k_0}{2} s^2 + o(s^2). \end{cases}$$

Therefore, the shape of the neighborhood of this point on curve $\vec{r}$ is approximately a parabola (see Fig. 4):

$$y = \frac{k_0}{2} x^2;$$

in other words, the curvature k_0 in this neighborhood is approximately

$$k_0 = \frac{2y}{x^2},$$

since k_0 is proportional to $\frac{y}{x^2}$, we can compare the corresponding value of $\frac{y}{x^2}$ when we need to compare the curvature at different points. In other words, if we use

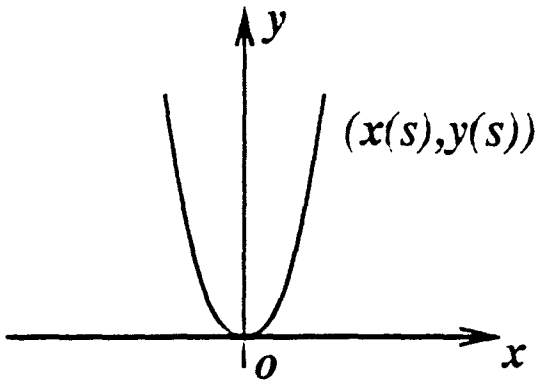


Fig.4. The shape of the neighborhood of a point is approximated as a parabola.

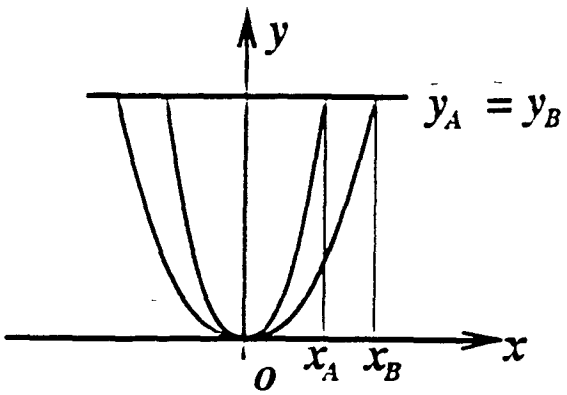


Fig.5. Curvature comparison of two curves.

(x_A, y_A) and (x_B, y_B) to represent the local coordinates at points A and B, respectively, we can see that the curvature at point A is greater than that at point B if $\frac{y_A}{x_A^2} > \frac{y_B}{x_B^2}$. Especially when $y_A = y_B, x_A < x_B$ if $k_A > k_B$. That is, the parabola at A is steeper than that at point B. If we move the curve segment L_B at point B to point A, L_A should be above L_B (see Fig. 5).

The above statement can also be proved regorously as follows: Let the equations of curve L_A and L_B are $\mathbf{r}_A(s) = (x_A(s), y_A(s))$ and $\mathbf{r}_B(s) = (x_B(s), y_B(s))$, respectively. Assume that they both pass the origin at $s = 0$, that is, $\mathbf{r}_A(0) = \mathbf{r}_B(0) = 0$ (= (0,0)). Using $k_A(s)$ and $k_B(s)$ to represent the curvature of curve L_A and L_B, respectively, when both $k_A(s)$ and $k_B(s)$ are positive, the curve L_A and L_B at $s = 0$ are approximately parabolic. We now assume that the curvature of L_A at $s=0$ is greater than the curvature of L_B at s_0, i.e., $k_A(0) > k_B(0)$, then when both L_A and L_B are smooth, by continuity, we should have, in the neighborhood of $s=0$,

$$k_A(s) > k_B(s), \tag{2}$$

If $\theta_A(s)$ and $\theta_B(s)$ represent the angle between the tangent lines of curves L_A and L_B, respectively, and the x-axis, then by the taking of coordinate system we have $\theta_A(0) = \theta_B(0) = 0$ and $0 \le \theta_A(s), \theta_B(s) \le \frac{\pi}{2}$. Consequently, from the definition of curva-

ture $k(s)$, we obtain:

$$\int_0^s k(\sigma)d\sigma = \int_0^s \frac{d\theta(\sigma)}{d\sigma}d\sigma = \theta(s)-\theta(0) = \theta(s).$$

Therefore, when s is small enough, we can obtain from *Eq.* (1):

$$\theta_A(s) = \int_0^s k_A(\sigma)d\sigma > \int_0^s k_B(\sigma)d\sigma = \theta_B(s). \tag{3}$$

This equation indicates that, in the neighborhood of origin, the slope of the tangent line at curve L_A is greater than that of L_B. From *Eq.* (2), $\frac{dx}{ds} = \cos\theta$, $\frac{dy}{d\theta} = \sin\theta$ and the assumption that $x_A(0) = x_B(0) = y_A(0) = y_B(0) = 0$, we obtain

$$x_A(s) = \int_0^s \cos\theta_A(\sigma)d\sigma < \int_0^s \cos\theta_B(\sigma)d\sigma = x_B(s).$$

$$y_A(s) = \int_0^s \sin\theta_A(\sigma)d\sigma > \int_0^s \sin\theta_B(\sigma)d\sigma = y_B(s).$$

Specially if $y_a(s) = y_B(s)$ (see *Fig.* 6), then

$$\int_0^{s_2} \sin\theta_B(\sigma)d\sigma = y_B(s_2) = y_A(s_1) = \int_0^{s_1} \sin\theta_A(\sigma)d\sigma > \int_0^{s_{s_1}} \sin\theta_B(\sigma)d\sigma.$$

Hence, $s_2 > s_1$. Further, we have

$$x_A(s_1) = \int_0^{s_1} \cos\theta_A(\sigma)d\sigma < \int_0^{s_2} \cos\theta_A(\sigma)d\sigma < \int_0^{s_2} \cos\theta B(\sigma)d\sigma = x_B(s_2).$$

That is:

$$x_A(s_1) < x_B(s_2). \tag{4}$$

Eq. (3) and the assumption $y_A(s_1) = y_B(s_2)$ lead to the following result:

$$\frac{y_A(s_1)}{x_A^2(s_1)} > \frac{y_B(s_2)}{x_B^2(s_2)}.$$

Eq. (3) shows that, if the coordinate is taken from neighborhood of $s = s_A$ on curve L_A and of $s = s_B$ on curve L_B, then when the coordinates are the same, the abycissa of the curve with the greater curveture is smaller than that of smaller curvature. In short, we can use $\frac{y}{x^2}$ as the measurement of the curvature in the neighborhood of a point of the curve, where x, y are the local coordinates in the neighborhood of that point.

From the above discussion, we see that if for a candidate point P_i its value of k_i exceeds some threshold, then it could be considered as a corner point. Otherwise we could eliminate it. Usually this threshold is task-dependent.

3.3 The Algorithm

We now embed the above ideas into our algorithm of corner points detection along the whole image boundary. To make the method work, we must first divide the whole image boundary into several segments. Usually the number of points on a segment is taken as a sixth or eighth of the total number of points on the whole boundary. However, some problem may occue if a corner point is just a joint point of two segments by our segment selection. Fig. 6(a) shows such an example. In this case, point P is a corner point, but it is ignored because of the segment selection. The problem could be solved by letting the segments be overlapped to some extend. In our experiment, we took the extend of the overlapped area as a half or a third of the segment. The method is shown in Fig. 6(b). According to the above discussion, we briefly list our algorithm as follows:

1. Set the search boundary segment length to $\frac{1}{8}$ of the whole boundary;
2. Select the first boundary segment;
3. **while** the search area not cover the whole boundary **do**

3.1. **For** each of the segments **do**

3.1.1. Compute the distances from the boundary points of the segment to the straight line connecting the end points of the segment;

3.1.2. Select the points which are the local maximum or local minimum as the corner point candidates.

3.1.3. **For** each of the candidates **do**

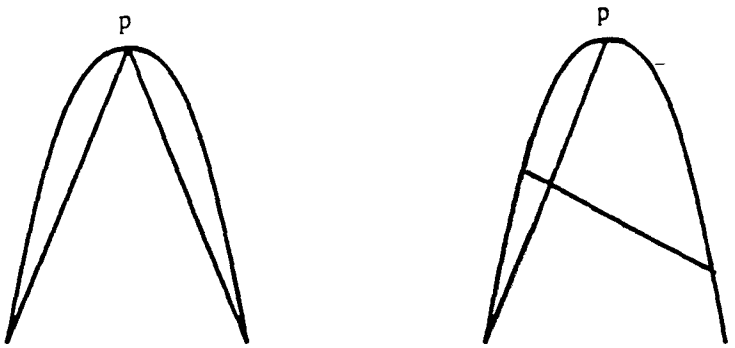
3.1.3.1 Calculate the curvature characteristical value K_i according to formula (1);

3.1.3.2 **If** $K_i >$ threshold, then accept the candidate as a corner point; Otherwise reject it.

3.2. Advance the search segment by half length of the segment;

3.3. **endwhile**

Because of the presence of noises, it may be true that sometimes the place where our candidate located is not the place where the real one is located. But



(a) A case in which a corner may be ignored.

(b) The method to overcome the problem.

Fig.6. An example of failure detection of a feature point and the solution to it.

according to some statistical data, we can safely assert that the true corner point will be in a neighbor area of the candidate if this area contains one. Thus in step 3.1.3.2 of the above algorithm, instead of just calculating the curvature characteristical value of only the candidate, we could modify it so that it calculates every point in the neighbor area where the candidate locates. By select the point with maximum curveture characteristical value among the points ine the area, we could thus avoid the deviation of the corner points.

4. Implementation of the Algorithm

4.1. Candidate Points Selection

Assume that we are given a segment of image boundary which is composed of points $P_0(x_0, y_0), P_1(x_1, y_1), ..., P_k(x_k, y_k)$ represented in Cartesian coordinate system. First we make a chord connecting the end points of this segment, that is, the straight line segment passing through $P_0(x_0, y_0)$ and $P_k(x_k, y_k)$. The line equation for this chord is:

$$y - y_0 = \frac{y_k - y_0}{x_k - x_0} (x - x_0). \tag{5}$$

Or, by some processing, we could write its general form:

$$Ax + By + C = 0, \tag{6}$$

where A, B, C are constants which can be represented as the combination of x_0, y_0, x_k, y_k. Then according to analytical geometry, the distance from a point $P_i(x_i, y_i)$ on the curve to this line could be written as:

$$d_i = \frac{|Ax_i + By_i + C|}{\sqrt{A^2 + B^2}}, \tag{7}$$

or more precisely:

$$d_i = \frac{|(y_i - y_0)(x_k - x_0) - (y_k - y_0)(x_i - x_0)|}{\Delta} \tag{7'}$$

where $\Delta = \sqrt{(y_k - y_0)^2 + (x_k - x_0)^2}$ is the length of the chord.

After the distances from all of the points on the curve to the chord $\overline{P_0P_k}$ have been calculated, we can then select the local maxima or local minima by comparing these distances. Assume that P_m is a local maximum in a neighbor area from P_{m-l} to P_{m+r}, where l and r are the length of left and right area respectively, then we have:

$$d_m > d_i \quad \text{for } m-l \le i \le m+r, i \ne m,$$

where d_i and d_m are the distances from points P_m and P_i to the chord respectively. Or at least we have:

$$d_m > d_{m-1}, \text{ and } d_m > d_{m+1} \tag{8}$$

respectively. If we define a sign function as

$$sign(P_m) = \begin{cases} 1, & \text{if } d_m - d_{m-1} > 0 \\ 0, & \text{if } d_m - d_{m-1} = 0 \\ -1, & \text{if } d_m - d_{m-1} < 0 \end{cases}.$$

We can pick up points which are satisfied by Eq.(8) in a very simple way. For a point P_m to satisfy the Eq.(8), it is clear that:

$$sign(P_m)=1, \text{ and } sign(P_{m+1})=-1,$$

that is:

$$sign(P_m)\cdot sign(P_{m+1})=-1, \tag{9}$$

and

$$sign(P_m)>0. \tag{10}$$

Similarly, for a point $P_{m'}$ to be a candidate of local minimum, we would have:

$$d_{m'} < d_{m'-1}, \text{ and } d_{m'} < d_{m'+1}. \tag{11}$$

By similar development, we get the condition for local minimum candidates, which is:

$$sign(P_{m'}) \cdot sign(P_{m'+1}) = -1, \tag{12}$$

and

$$sign(P_{m'}) < 0. \tag{13}$$

This mechanism could be implemented in following steps:

```
sign1:=1;  d1:=0;
for i:=1 to k-1 do
begin
    compute d2: the distance from P_i to the chord according to Eq.(7');
    if d2-d1>0 then
        sign2:=1  else
            if d2-d1=0 then
                sign2:=0
            else
                sign2:=-1;
    if sign1*sign2=-1 then
        if sign1>0 then
            mark P_i as a local maximal candidate
        else
            mark P_i as a local minimal candidate;
    sign1:=sign2;  d1:=d2;
end;
```

By study formula (7) and (7') further, we could find that there is a minor problem: Because the distance d_i calculated from Eq.(7) or (7') is an absolute value, then we would have such cases shown in Fig.7 which satisfy the satisfy the conditions (9), (10), or (12), (13). But in these cases they are neither the local maximum nor the local minimum.

To overcome this problem, we introduce the concept of *virtual distances*. Consider a straight line in the *XOY* plain whose equation is:

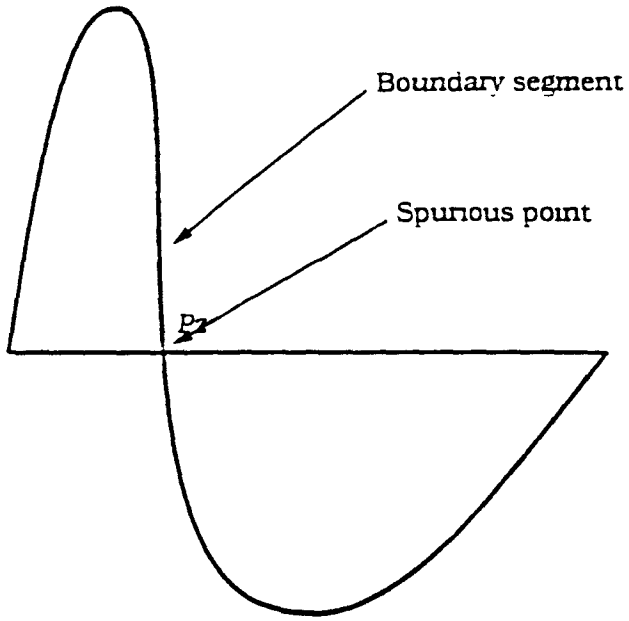


Fig.7. An example of detection of a spurious point.

$$Ax + By + C = 0.$$

Assume $P_i(x_i, y_i)$ is a point on XOY plain. From the analytical geometry we know that if P_i is on the line, then we have:

$$Ax_i + By_i + C = 0.$$

If $P_i(x_i, y_i)$ is above the line, then we have:

$$Ax_i + By_i + C > 0.$$

and finally, if $P_i(x_i, y_i)$ is under the line. we have:

$$Ax_i + By_i + C < 0.$$

From this point of view. we define a virtual distance from a point to a straight as follows: First we calculate the distance from P_i to the line according to Eq(7) or (7'). The we check the position of P_i related to the line. If P_i is above the line, we make

this distance positive: if P_i is under the line, we make it negative. In fact, it is a very easy task to calculate the virtual distances by remove the absolute operation from Eq.(7) or (7'). Thus the formula to find a virtual distance from point P_i to a line is simply:

$$d_i^v = \frac{Ax_i + By_i + C}{\sqrt{A^2 + B^2}} \tag{14}$$

or

$$d_i^v = \frac{(y_i - y_0)(x_k - x_0) - (y_k - y_0)(x_i - x_0)}{\sqrt{(y_k - y_0)^2 + (x_k - x_0)^2}} \tag{14'}$$

To show that the virtual distances can help us solving the problem, consider the situation shown in Fig.7. Assume that the boundary segment cross with the chord at point P_z, with its left neighbors $P_{z-1}, P_{z-2}, \ldots$, above the chord and its right neighbors $P_{z+1}, P_{z+2}, \ldots$, under the chord. Clearly we have $d_z^v = 0$. All of its left neighbors, according to the definition of virtual distance, have the positive virtual distances, and all of its right neighbors will have negative distances. Furthermore, we can also arrange these virtual distances in a decreasing order like this:

$$\ldots > d_{z-2}^v > d_{z-1}^v > d_z^v = 0 > d_{z+1}^v > d_{z+2}^v > \ldots$$

Thus, at point P_z, we have:

$$sign(P_z) = sign(d_z^v - d_{z-1}^v) = -1,$$

$$sign(P_{z+1}) = sign(d_{z+1}^v - d_z^v) = -1,$$

and

$$sign(P_z) \cdot sign(P_{z+1}) = (-1)(-1) = 1,$$

which means that P_z could not be picked up as a candidate of local maximum or local minimum. The above algorithm can be modified to implement this: instead of calculating distance d_i according to Eq(7'), we calculate the virtual distance d_i^v according to Eq(14').

4.2. Candidate points Determination

After all of the candidates of possible local maximum or local minimum have been detected, we should further decide which points are most promising candidates of corner points. Because of the presence of noises, it is not a easy task to differentiate a real candidate from a spurious one. Fig.8 shows some cases that results in the generation of spurious candidates.

In Fig.8 point P_2 is a probable noisy peak and it has been picked up as candidate because it satisfies the condition. This kind of spurious points are relatively easy to eliminate. Because the range of this kind of noisy peak is usually only a few points, by setting up a threshold on the range, they could usually be eliminated. The usual way to eliminate a spurious point is to calculate the length of left and right arm of a candidate point. If both the left and the right arm lengthes exceed a threshold and the product of them (sometimes we use the product of the values of some function of them, such as square root, or logrithm, etc.) also exceeds a threshold, then it is considered as a more promising candidate. To express this idea more precisely, assume that the lengthes of left and right arms of a candidate point P_i is l and r respectively, then the decision rule for P_i is:

1. l, r are both greater than a threshold;
2. $f(l)\cdot f(r)$ is greater than a threshold. (Here the function f may take the form of: $f(x)=\sqrt{x}$, $f(x)=x$, *or* $f(x)=lnx$).

Then the question is: how can we determine the length of left or right arm for a given candidate point P_i? One way is that we count the number of points from P_i to the left or right with continuously increasing or decreasing order the virtual distances, that is, we select r such that:

$$d_i^v > d_{i+1}^v > \ldots > d_{i+r}^v < d_{i+r+1}^v , \tag{15}$$

or

$$d_i^v < d_{i+1}^v < \ldots < d_{i+r}^v > d_{i+r+1}^v , \tag{15'}$$

as the length of its right arm. The length of left arm can be selected in a similar way.

There is subtle case which can cause problem for this decision rule. Consider a situation in Fig.9: Here P is a real candidate, P' is a noise. The actual end of the right arm of P is P''. But because of the existance of noise P', it may cause that the right arm length calculated by the above decision rule with only few points. And if this length is under the value of threshold, it may also be eliminated. Thus causes a failure of detection.

To solve this kind of problem, we use the following mechanism: When we meet a point P_{i+k} which violates the condition (15), that is, say:

$$d_{i+k-1}^v < d_{i+k}^v > d_{i+k+1}^v ,$$

we count from P_{i+k} the number of points which are violate this condition until either at some point the condition is restored or the number of violating points has

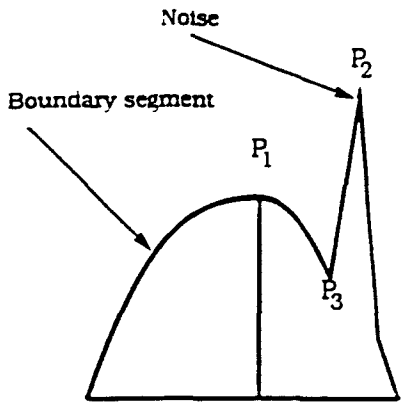


Fig.8. A boundary segment with presence of noise.

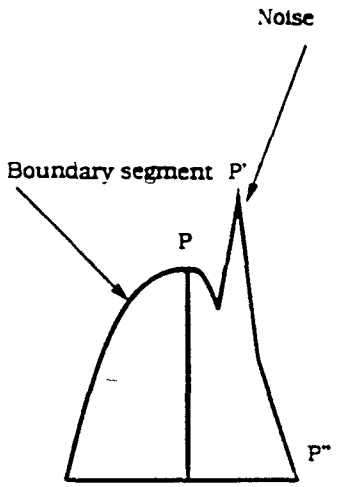


Fig.9. A case when the noise is very close to the feature point.

exceeded some threshold. For the former case, we consider these violating points as a noise and just ignore them and continue the process from the restoring point. For the latter case, we don't think them as a noise, but as a part of another candidate. In this case, the actual length of the right arm we found is k. (The mechanism has been tested separately with a set of test data. The testing data set and the description of the results is listed in Appendix II)

After the left arm and the right arm have been found in this way, we then consider the value of the square root of the product of the arms, it is:

$$k_i = \sqrt{l_1 l_2}$$

where l_1 is the length of the left arm of the candidate point P_i and l_2 is length of its right arm. According to this k_i, we can eliminate some spurious points. If k_i is less than a threshold, then we consider it as a spurious candidate and just eliminate it. Otherwise we accept it for further consideration. The reason that we use the square root of the product of the length of the arm is that: The length of the arms have no doubt the contribution for a candidate to be a corner, but this contribution is not

proportional to the length of the arms.

We can now summarize the above discussion in following steps:

Initialize the noise count;
Initialize the left and right arm counts;
for each candidate P_k **do**
    find the left arm of P_k:
    Repeat
        i←k-1;
        while P_i violate the arm condition **and** noise count less than threshold **do:**
            increment the noise count;
            decrement i;
        if noise count exceeds the threshold **then** exit;
        increment the left arm count;
        decrement i;
    forever
    find the right arm of P_i:
    Repeat
        i:=k+1;
        initialize the noise count to 0;
        while P_i violate the right arm condition and noise count less than threshold **do**
            increment the noise count;
            increment i;
        if noise count exceeds the threshold **then** exit;
        increment the right arm count;
        invrement i;
    forever Decide whether accept or reject P_k:
        if sqrt(leftcount*rightcount)>threshold **then**
            accept P_k for further consideration;
        else
            eliminate P_k;
    endfor_

4.3. Final Selection of the Corner Point

For those candidates survived from the elimination steps described in Sec. 4.2, we must further consider the change of curvature around these points. Consider an arc segment in the neighbor of a candidate point P_i as shown in Fig.10 We denote the span of this arc as x, and the height as y. By the conclusion of Section 3.2, we can determine that: if the ratio of y over the square of x is high, this also means that the point P_i has high curvature. In other words, the curvature c_i at the point P_i is proportional to this ratio:

$$c_i \propto \frac{y}{x^2}.$$

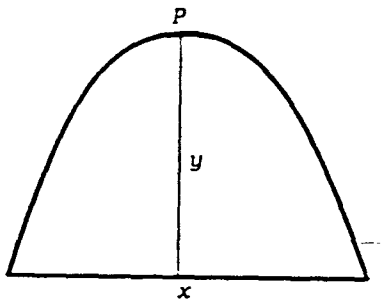


Fig. 10. The geometrical meaning of Formula (1).

We can use this formula as a decision rule, that is: if c_i is greater than a given thrshold, then the curvature at P_i is large enough for P_i to be picked up as a corner point, otherwise we reject P_i as a corner point.

Before we implement this mechanism, we must take little further consideration: There may be cases in which the candidate is biased little from the real corner point. To find more precise position of the corner point, we could test the curvatures of all the points in a neighbor area of the candidate, and select the the point with the highest curvature among the points in this area as the corner point. In this way, the location of the corner would be more precise.

We describe the algorithm as follows:

Algorithm: Final Detection of Corner Point;
Input: A corner Point Candidate P_i;
Output: P'_i in a neighbor area of P_i with the maximal value of y/x^2 and this value exceeds a given threshold, or reject P_i if no such a point found.

1. Pick up δ as the range of the neighbor area;
   max_c←0; max_loc←0;
2. **for** j←i - δ to i + δ **do**
   y_j ← height of the arc centered at P_j;
   x_j ← span of the arc centered at P_j;
   $c_j \leftarrow y_j/x_j^2$;
   if $c_j > max\_c$ **then** max_loc←j;
   end;
   (* Now the point with the maximal curvature has been determined. Next step is to determine whether we should accept it or reject it *)
3. **if** max_c > threshold **then**
   output P_{max_loc} as a corner point;
   else
   reject it;

end of the algorithm

5. Experiments Results

The method described in this paper is implemented in C language running on the Sun system. Following are some experiments used to test its performance.

In the figure of Fig. 11, the seven corner points are detected with the value of threshold of the curvature characteristical value equal to 0.08. From the figure we see that the location of the corners are detected precisely. Fig. 12 also shows a very successful detection. In Fig. 13, most of the corner points are detected correctly. Point x should be a corner point, but the method fails to detect it. The method also failed to detect the point y, but this is a very confusing point. The feature of this point is not very sharp. In fig. 14 all of the interesting feature points are detected. But it also generated two spurious points (x and x'). The boundary in Fig. 15 is a subtle case. In this figure, two seemed interesting points was not detected. By changing some value of thresholds, they could be detected. But it may also generate some points which we

don't want. It seemed that for the complicated image boundary some further study of the relationship between the detectability and the value of parameters is still needed.

The experiments also show that the method is very efficient. By analysis we know that the time complexity for the method is $O(n)$, where n is the total points of the image boundary.

6. Conclusion

In this paper, a heuristical method mainly based on the concept of SBD and the curvature characteristical value is discussed. Some experiments show that, although some further work for the refinement of the method is needed, it works for a variety set of image boundary. Some mechanisms dealing with the noises on the boundary are also proposed to improve the precision of the method. One of the advantages of the method is its simplicity. Also it is a very efficient method.

Acknowledgement

The author would like to thank Dr. David T. Wang who advised this research work. Without his instructions it is impossible that this work is done. The author would also like to thank the visiting professor Guizhang Tu who helped the author in solving the mathematical problem of this research work.

References

1. Haruo Asada, Michael Brady, "The Curvature Primal Sketch", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No.1, Jan. 1981, pp 2-14.
2. Liu H. C., Srinath M. D., "Corner Detection From Chain-Code", Pattern Recognition, Vol. 23, No.1/2, pp 51-68, 1990.
3. Lawrence O'Gorman, "Curvilinear Feature Detection from Curvature Estimation", 9-th ICPR, Rome, Nov. 1988, pp 1116-1119.
4. Lawrence O'Gorman, "An Analysis of Feature Detectability from Curvature Estimation", CVPR, Ann Arbor, Jun. 1988, pp 235-240.
5. Gerard Medioni, Yoshio Yasumoto, "Corner Detection and Curve Representation Using Cubic B-splines", Computer Vision, Graphics, And Image Processing 39, pp 267-278, 1987.
6. Azriel Rosenfeld, Joan S. Weszka, "An Improved Method of Angle Detection on Digital Curves", IEEE Trans. Comput., Sept., pp 940-941, 1975

7. Beus H. L., Tiu S. S. H., "An Improved Corner Detection Algorithm Based on Chain-Coded Plane Curves", Pattern Recognition Vol. 20, No. 3, pp 291-296, 1987.

8. Rosenfeld A., Johnston E., "Angle Detection on Digital Curves", IEEE Trans. Comput., 22: pp 875-878, 1973.

9. Cheng F., Hsu W., "Parallel Algorithm for Corner Finding on Digital Curves", Pattern Recognition Lett. 8: 47-53, 1988.

10. Rosenfeld A., Thurston M., "Edge and Curve Detection for Visual Scene Analysis", IEEE Trans. Comput., Vol., c-20, No.5, pp 562-569, 1971.

11. Freeman H., Davis L. S., "A Corner Finding Algorithm for Chain-Coded Curves", IEEE Trans. Comput. 26: pp 297-303, 1977.

12. Wang D. T., Wei C.S., Chen S.S., Sung B.C., Shiau T.H., Ng P.A., "Cross Correlation of Sampled Boundary Distances - An Approach to Object Recognition", Proc 1st Int'l Conf System Intergration, Morristown, New Jersey, April 23-26, 1990.

Appendix I: Source Program Listing for the Implementation of the Method.

```
/*************************************************************/
/*
/*   This is a program of using the corner finding
/*   algorithm to find local features points.
/*
/*   Programmer : Qiulin Li
/*   Date       : Dec., 1991
/*
/*************************************************************/

#include <stdio.h>
#include <math.h>

#define MAXLEN          1500  /* the maximum points of image */
#define SQUARE(A)         ((A) * (A))
#define LEFT          5    /* maximum points of left arm */
#define SELECT         4    /* define interleave of point */
#define RIGHT          5    /* maximum points of right arm */
#define YES           1
#define ABS(a) (a >= 0 ? a : -a)
#define Noisethreshold 3
#define Armthreshold 5
int len;                  /* number of points of image */
int local;                /* number of points of local calculation */
struct node       {          int x;          int y;
       int n;
      } ;
struct sor{
         float length;
         int point;
        };
struct sor *height();     /* function to find the vertical distance */
```

```
struct node point[1000];  /* array storing the cordinate of boundary points */
struct node maxnmin[200];
struct sor array[100];
int RANGE;  /* global range */
int HOLD;   /* value of right*left
               right and left are defined in the program */
float c_hold: /* value of y/(x*x) */
int Feature_points[100], nxt_feature_pt=0;
void insert();
void output_feature_points();
int get_sign();


main ()
{
 int i,n;
 FILE *fp,*fpp;
 char boundaryfile[20];
 char *argv[2];

 /* input image file name. filename.bd */

 printf(" Enter filename for the image boundary : ");
 scanf("%s",boundaryfile);
 fpp = fopen(boundaryfile,"r");
 while(fpp == NULL)
   {
    printf("*** Open error, no such raw image file\n");
    printf("    Rekey-i imageboundary file name please : ");
    scanf("%s", boundaryfile);
    fpp = fopen(boundaryfile,"r");
   }
                              /* get input from image file */
 get_input (&len,fpp);
                              /* input global range, EVEN number */
 RANGE=len/8;
```

```
 c_hold=0.06;
 max_and_min(&len);
 output_feature_points();
 fclose(fpp);
 }

/*
   Read image boundary datas and boundary length.
*/

get_input(len,fp )

int *len;
FILE *fp;
{
int i;

 fscanf(fp,"%d",len);
 for ( i = 0; i < *len  ; i++ )
 {
   point[i].n = i;
   fscanf(fp,"%d %d",&point[i].y,&point[i].x);
 }
}

/*************************************************************************/

/*
   find local feature points
*/

max_and_min(leng)
int *leng;
{
 int i,j,count,from_zero;
 float lon;
```

```
struct sor *max_array;

for(i=0;i<*leng;)
{
   from_zero = 0;
   count = i;
   for(j=0;j<RANGE;j++) /* read points in the range into array */
   {
    if(count < *leng)
    {
       maxnmin[j].x = point[count].x;
       maxnmin[j].y = point[count].y;
       maxnmin[j].n = point[count++].n;
    }
    else                    /* pass last point */
    {
       maxnmin[j].x = point[from_zero].x;
       maxnmin[j].y = point[from_zero].y;
       maxnmin[j].n = point[from_zero++].n;
    }
     i++;
   }
   max_array = height(maxnmin,&lon,RANGE); /* array storing the vertivcal
distance */
   trim(max_array,lon); /* first trim determined by right*left */
   i = i - RANGE/SELECT;
 }
}

/*****************************************************************/

/*
    calculate the vertical distance
*/

struct sor *height(cmaxnmin,l,range)
```

```
struct node *cmaxnmin; /* array of the points in the range */
int range; /* local max calculation range */
float *l;  /* distance between the first point and the last point */
{
 float raw_area,distance,area,fullarea;
 float verticle;
 float xx,yy;
 int index,i,max_lo_ind;
 float x1,y1,x2,y2,slope;
 int inverse,sign;

 x1=cmaxnmin[0].x;        y1=cmaxnmin[0].y;
 x2=cmaxnmin[range-1].x;  y2=cmaxnmin[range-1].y;
 if (x1==x2)
 {
  inverse=1;
 }
 else
 {
  inverse=0;
  slope=(y2-y1)/(x2-x1);
 }
 index = 0;
 xx = SQUARE((cmaxnmin[range-1].x - cmaxnmin[0].x));
 yy = SQUARE((cmaxnmin[range-1].y - cmaxnmin[0].y));
 distance = sqrt((double)(xx + yy));
   /* distance between the first point and the last point */
 for(i=0;i<range;i++)
 {

  /* area of triangle divided by distance to get the vertical distance */

   raw_area = (cmaxnmin[i].y-cmaxnmin[0].y)
            *(cmaxnmin[range-1].x-cmaxnmin[0].x)
            -(cmaxnmin[range-1].y-cmaxnmin[0].y)
            *(cmaxnmin[i].x-cmaxnmin[0].x);
```

```
    fullarea = ABS(raw_area);
    verticle =   fullarea / distance;
    if (inverse)
      sign=cmaxnmin[i].x-x1;
    else
      sign=cmaxnmin[i].y-y1-slope*(cmaxnmin[i].x-x1);
    if (sign>0)   sign=1;
    else
    {
      if (sign<0) sign=-1;
    }
    array[index].length = verticle*sign;
    array[index++].point = cmaxnmin[i].n;
  }  /* end */
  *l = distance;
  return array;

}
/*****************************************************************/

/*
   find loacl maximum determined by  right*left
*/

trim(rarray,d)
struct sor *rarray;
float d;
{
  int i,max_ind,j,k,left,right;
  int left_noise,right_noise,left_limit,right_limit;

typedef struct {
   int del_bit;
   int what;
   int leftarm,rightarm;
   int point_inx;
```

```
  } candidate;
candidate cand_ary[100];
int cand_ptr=0;
int sgn1,sgn2;
int stop,noise;
float stop_size;


/* find all of the candidates of local maxima or local minima */
sgn1 = get_sign(rarray[0].length,rarray[1].length);
for (i=1; i<RANGE-1; i++)
{
  sgn2=get_sign(rarray[i].length,rarray[i+1].length);
  if (sgn1*sgn2<0)
  {       cand_ary[cand_ptr].del_bit=0;       cand_ary[cand_ptr].point_inx=i;
    cand_ary[cand_ptr++].what=(sgn1>0?1:0);
  }
  sgn1=sgn2;
}

 left_limit=0;
 right_limit=RANGE-1;

/* noise elimination */
/* for every candidate do */
 for (k=0; k<cand_ptr;k++)
 {
   stop=0;
   left=right=0;
   /* Find the left arm */
   j=cand_ary[k].point_inx;
   if (cand_ary[k].what==1)
   {
     while (j>left_limit)
     {
       if (rarray[j-1].length>rarray[j].length)
```

```
      {   /* a possible  noise */
        stop_size=rarray[j].length;
        j--;    noise=0;  stop=0;
        while ((rarray[j].length>stop_size) & ((j--)>left_limit))
           noise++;
        if (noise>Noisethreshold)
           stop=1;
      }
      else
      {
        left++;   j--;
      }
      if (stop) break;
    }  /* while */
  } /* if */
  else
  {
    while (j>left_limit)
    {
      if ( rarray[j-1].length<rarray[j].length )
      {
         stop_size=rarray[j].length;
         j--;   noise=0;    stop=0;
         while ((rarray[j].length<stop_size) & ((j--)>left_limit))
            noise++;
         if (noise>Noisethreshold)
            stop=1;
      }
      else
      {
         left++;   j--;
      }
      if (stop) break;
    } /* while */
  } /* else */
  /* Find the right arm */
```

```
j=cand_ary[k].point_inx;
stop=0;
if (cand_ary[k].what==1)
{
  while (j<right_limit)
  {
    if (rarray[j+1].length>rarray[j].length)
    {
       stop_size=rarray[j].length;
       j++;    noise=0;    stop=0;
       while ((rarray[j].length>stop_size) & ((j++)<right_limit))
          noise++;
       if (noise>Noisethreshold)
          stop=1;
    }
    else
    {
       right++;  j++;
    }
    if (stop)  break;
  } /* while */
} /* if */
else
{
  while (j<right_limit)
  {
    if (rarray[j+1].length<rarray[j].length)
    {
       stop_size=rarray[j].length;
       j++;      noise=0;    stop=0;
       while ((rarray[j].length<stop_size) & ((j++)<right_limit))
          noise++;
       if (noise>Noisethreshold)
          stop=1;
    }
    else
```

```
            {
              right++;  j++;
            }
            if (stop) break;
          }
        }
        if ((sqrt((double) left)*sqrt((double) right))<Armthreshold)
          cand_ary[k].del_bit=1;
        else
        {
          if ((left<LEFT)||(right<RIGHT))
            cand_ary[k].del_bit=1;
          else
          {
            cand_ary[k].leftarm=left;
            cand_ary[k].rightarm=right;
          }
        }
      }

      for (i=0; i<cand_ptr; i++)
      {
        if (cand_ary[i].del_bit==0)
        {
          local=min(cand_ary[i].leftarm,cand_ary[i].rightarm);
          if (local>5)  local=5;
          curve(rarray[cand_ary[i].point_inx].point);
        }
      }
      return;
    }

    /*****************************************************************/
    int get_sign(x1,x2)
    float x1,x2;
    {
```

```
  float t;
  int sign;

  t=x2-x1;
  if (t==0)  sign=0;
  else
  {
    if (t<0)  sign=-1;
    else sign=1;
  }
  return sign;
}


/*
     calculation in local range
*/

curve(max)
int max;
{
int try,start,end.max_point;
float curvature.max_curvature,xsquare,vertical;

  max_curvature=0;    max_point=0;
  for (try=max-6; try<=max+6; try++)
  {
    start=try-local;
    if (start<0)  start+=len;
    end=(max+local)%len;
    xsquare=SQUARE(point[end].x-point[start].x)
          +SQUARE(point[end].y-point[start].y);
    vertical=(point[try].y-point[start].y)*
          (point[end].x-point[start].x)-
          (point[end].y-point[start].y)*
          (point[try].x-point[start].x);
```

```
    vertical=ABS(vertical);
    curvature=vertical/xsquare;
    if (curvature>max_curvature)
    {
      max_curvature=curvature;
      max_point=try;
    }
  }
  if(max_curvature > c_hold)
     insert(point[max_point].n,&nxt_feature_pt);
}

int min(x,y)
int x,y;
{
  int result;

  if (x<=y)   result=x;
  else        result=y;
  return result;
}

void insert(point,ptr)
int   point,*ptr;
{
 int i=0;

 Feature_points[*ptr]=point;
 while (Feature_points[i++]!=point);
 if (i==*ptr+1)  (*ptr)++;
 return;
}


void output_feature_points()
/* Output the feature points together with the original boundary file into a file
```

```
*/
 {
 FILE *f;
 int  i;
 char fn[12];

    printf("Please input the output file name:");
    scanf("%s",fn);
    f=fopen(fn,"w");
    fprintf(f,"%d\n",len);
    for (i=0; i<len; i++)
    {
      fprintf(f,"%d %d\n",point[i].y,point[i].x);
    }
    fprintf(f,"\n%d\n\n",nxt_feature_pt);
    for (i=0; i<nxt_feature_pt;)
    {
      fprintf(f,"%d \n",Feature_points[i++]);
    }
    fclose(f);
    return;
 }
```

Appendix II: Data Set Used to Test the Mechanism of Noise Elimination Dicussed in Sec. 4.2.

Test 1: In the following segment, no noise is present. The only one corner point is Point P_{16}. The method can detect this point successfully. –

40 20 15 22 18 24 21 26 24 28 27 30 30 32 33 34 36 36 39 38 42 40 45 42 48 44 51 46 54 48 57 50 60 53 58 56 56 59 54 62 52 65 50 68 48 71 46 74 44 77 42 80 40 83 38 86 36 89 34 92 32 95 30 98 28 101 26 104 24 107 22 110 20 113 18 116 16 119 14 122 12

Test 2: In the following set, a noise generated two spurious corners. The real one is at point P_{16}, two spurious ones are P_{20} and P_{22}. In the test run, first the three points are detected as candidates. Then the spurious ones are eliminated by the mechnism. Following is the test data set.

40 20 15 22 18 24 21 26 24 28 27 30 30 32 33 34 36 36 39 38 42 40 45 42 48 44 51 46 54 48 57 50 60 53 58 56 56 59 54 62 52 65 54 68 62 71 58 74 50 77 44 80 40 83 38 86 36 89 34 92 32 95 30 98 28 101 26 104 24 107 22 110 20 113 18 116 16 119 14 122 12

Appendix III: Figures from the Experiments.

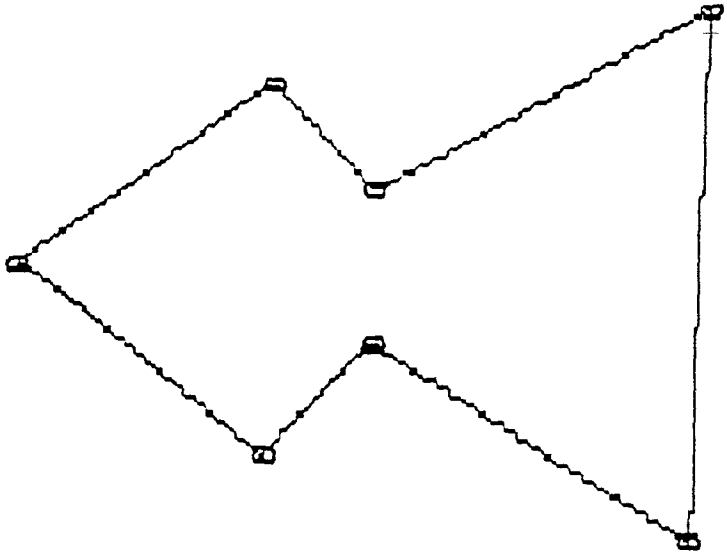


Fig.11. Experiment result 1.

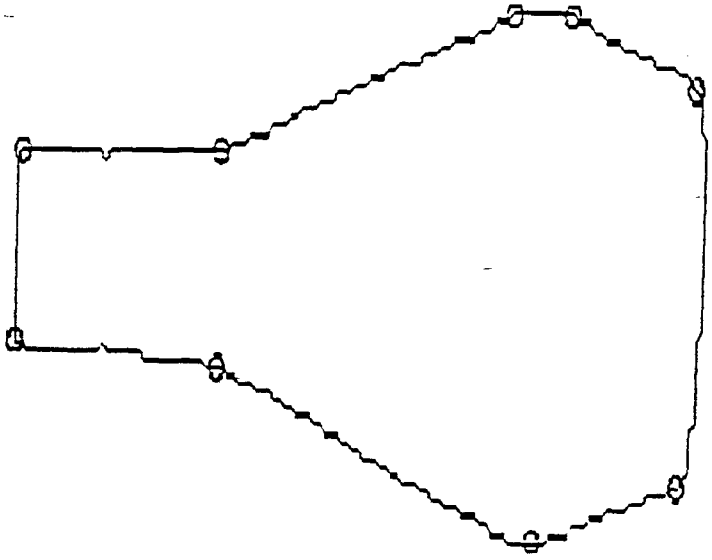


Fig.12. Experiment result 2.

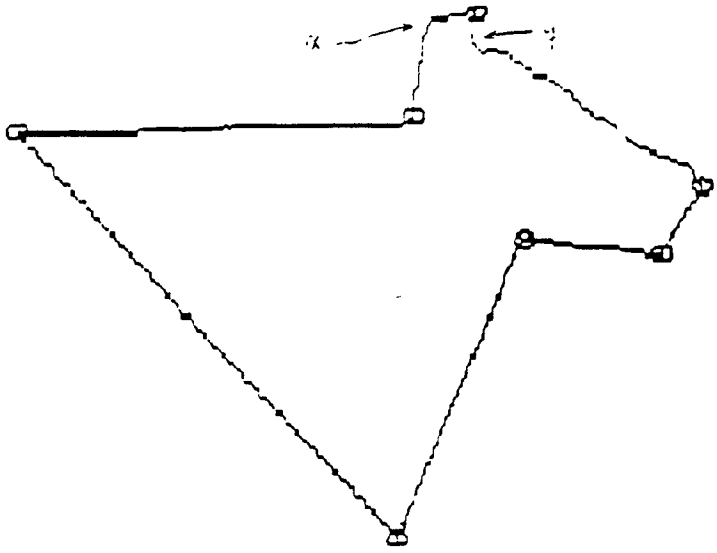


Fig.13. Experiment result 3.

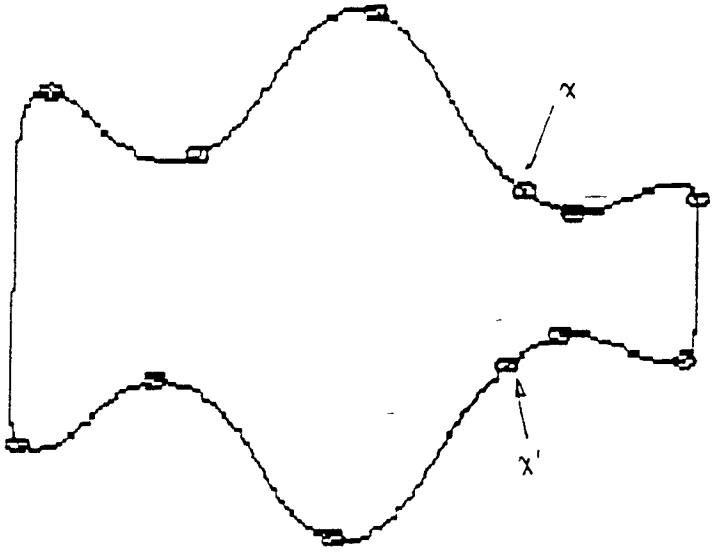


Fig.14. Experiment result 4.

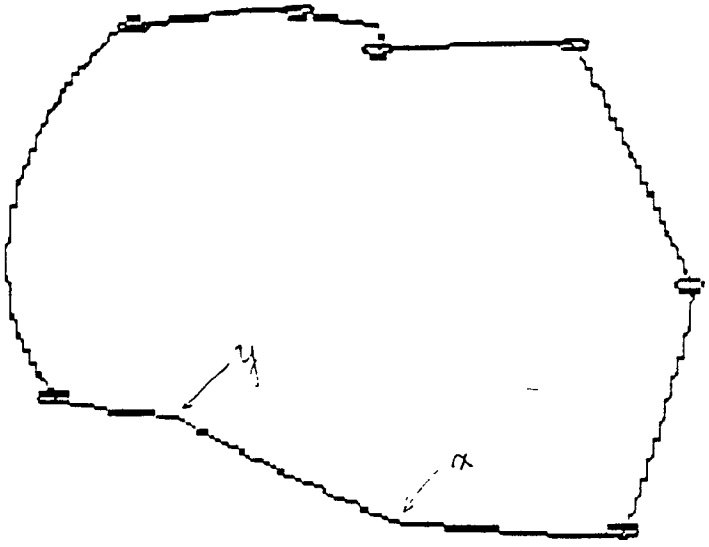


Fig. 15. Experiment result 5.