MVA'90 IAPR Workshop on Machine Vision Applications Nov. 28-30,1990, Tokyo

A Coarse to Fine Corner-Finding Method

Carlo ARCELLI† Andreas HELD‡ Keiichi ABE‡

† Istituto di Cibernetica, C.N.R., I-80072 Arco Felice, Naples, Italy
‡ Dept. of Computer Science, Shizuoka University, Hamamatsu, Japan

September 13, 1990

Abstract

An iterative algorithm for the detection of corner points on a digital curve is presented. The algorithm does not require any input parameters and yields intuitively appealing results for pictures varying in size and orientation. We first calculate the changes in curvature over a supporting arc whose length is either a fraction of the perimeter or given by the sub-arc bordered by two previously found corner points. We further scan the curvature changes, select all the pixels placed where the curve bends are locally significant, and finally mark maxima among the selected pixels as corner points. As the analysis is performed from a coarse to a fine resolution, features of different perceptual importance can be found and accordingly labelled. Some examples are shown in order to illustrate the performance of the algorithm.

1 Corner Points

An important step towards shape analysis and description is the representation of a shape by some simple and easy to handle elements. Suitable straight line segments, arranged so as to model the boundary of the shape, can be such elements.

The problem of finding the extremes of the segments has often been regarded as that of detecting the corner points on a digital curve, and several methods have been proposed to this aim. However, most of these methods are tuned to a certain application or a class of pictures, and are not ensured to yield really useful results with input pictures varying in size and orientation over a wide range (see [1,2] and the references therein).

In this paper, we present the first results obtained by using an algorithm which, essentially, does not require any input parameter and is to a certain extent rotation and size invariant.

Initially, boundary pixels are labelled with the curvature value obtained by computing the Gallus-Neurath code [3] over a supporting arc whose length is a fraction of the perimeter. Then, for every connected subset of the boundary, made up by pixels with label greater than the code corresponding to the perceptually significant minimal bend of the boundary, one pixel is appropriately selected as a candidate corner point at the coarsest resolution.

The process is repeated for every arc, delimited by a pair of candidate corner points successively encountered along the boundary. Since the Gallus-Neurath codes are computed over supporting arcs shorter than before, candidate corner points at finer resolution are found. The process continues through further subdivisions of the boundary, until no new candidate corner points can be detected. Finally, the set of the corner points is obtained from the set of the candidate corner points by removing those pixels which give only a marginal contribution to the perception of the global shape.

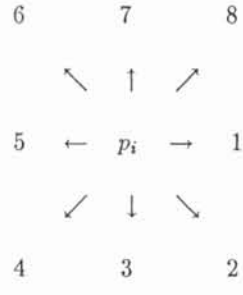
The corner points can be grouped into different levels, depending on the size of the supporting arc in correspondence of which they have been detected. Thus, as one moves from a low to a high resolution, some hierarchical knowledge about the shape is obtained. For both geometrical and blob-like figures, intuitively appealing results were achieved.

2 Fundamentals

Boundary-follower: Given a noise-free binary image, we scan the image from left to right, top to bottom, until we find a pixel belonging to the object and with at least one of its 4-connected neighbours belonging to the background. We mark this pixel as a boundary-point, and scan the eight neighbours counter-clockwise, starting at a background pixel , until the next boundary-point is found. Then, we move on in a similar way, until the origin is reached again. We further scan the image until we find another yet unmarked boundary-point, from where the same procedure is repeated. As a result,

outer boundary components are followed counterclockwise, whereas inner boundary components are followed clockwise, and an array of coordinates for each boundary component is obtained.

Chain code: The chain code [4] describes the spatial relationship of consecutive boundary-points. Given a starting point p_i, one out of a series of eight integers, $f = 1, 2, ..., 8$, indicates the direction in which the next adjacent boundary point lies. The relation between p_i and p_{i+1} is illustrated in the following figure :



Differential chain code: The differential chain code is defined as

$$c_i = f_i - f_{i-1} \tag{1}$$

and describes the relative change in direction occurring at point p_i. It is also mentioned as the 1-curvature at point p_i.

N-code: The N-code, or Gallus-Neurath code [3], is defined as

$$c_i^N = N c_i + \sum_{k=1}^{N-1} (N-k)(c_{i-k} + c_{i+k}) \tag{2}$$

where c_i is the 1-curvature at point p_i.

Each c_i^N measures the change in slope between lines (p_{i-N}, p_i) and (p_i, p_{i+N}), $c_i^N = N$ indicates a change of slope of approx. 45°, while $c_i^N = 2N$ an angle of approx. 90°, respectively.

The appropriateness of N-codes for evaluating curvature information has been shown in [5].

3 Method

For each boundary component, the coordinates of the boundary-points, the chain codes, the differential chain codes, and the N-codes are extracted and stored. As for the N-codes, they are computed with respect to a value of N chosen as

$$N = \max(\lceil P/8 \rceil, 3) \tag{3}$$

where P indicates the length of the boundary. This choice corresponds to a coarse approximation of a square to the shape, as we draw information from an arc of length P/8 on both sides of a point, i.e. from a supporting arc of length P/4.

At a coarse resolution, boundary points where the slope changes more than 45°, are assumed to be placed in correspondence with significant bends. Accordingly, the threshold

$$t_0 = \max(N, 2) \tag{4}$$

is chosen, and all the points with $\mid c_i^N \mid \geq t_0$ are marked.

Connected sets of marked points are regarded as intervals and, to avoid scattered points, an interval growing procedure is applied. Namely, for each interval we add on both sides the adjacent unmarked points for which $\mid c_i^N \mid \geq \max(t_0 - 1, 2)$. Then, in an additional scan of the N-chain, we delete all intervals of unit length and, in the remaining intervals, mark as peak the point where $\mid c_i^N \mid$ is a maximum. If in one interval there are more than one point with the same maximum $\mid c_i^N \mid$, the middle point is marked.

Recursively, the N-codes are computed for the substrings between two successive peaks. The value for N is still given by formula (3), where P is now the length of the substring. However, differing from before, we do not compute the N-codes in correspondence to the N-1 points placed at each end of the substrings, so as to diminish the influence of previously found peaks. Moreover, in order to detect smoother details of the boundary with any further iteration, the threshold decreases with the depth of the iteration, and is given by

$$t_l = \lfloor N \frac{6-l}{6} \rfloor; \quad l = 1, \ldots, 5 \tag{5}$$

Also, unit length intervals are no longer discarded.

If all the found peaks were taken as corner points, this procedure is likely to create a number of redundant points. Thus, we accept as corner points only those peaks surviving a triangulation process, which does not significantly alter the way the sequence of resulting segments approximates the input boundary.

For each triple $\{p_a, p_b, p_c\}$ of peaks, we compute the distance d from p_b to the straight line through p_a and p_c. If $d \geq 1$, we mark p_b as a corner point and move to the next triple $\{p_a, p_c, p_{c'}\}$. If $d < 1$, we move p_c to the next peak, $p_{c'}$, and let p_b take the position of all the peaks between p_a and $p_{c'}$, For all these triples we calculate d, and if $\max(d_1, d_2, ..) \geq 1$ we mark the peak with $\max(d)$ as corner point. Then we move on to the next triple. This procedure is repeated until the starting point is reached again.

Finally, the polygon approximating the boundary is created by joining pairs of corner points, as they are encountered when following the boundary.

4 Experimental Results

The algorithm has been tested with several shapes, and the pointwise error e_i between the boundary and the approximating polygon has been adopted to evaluate the results. The following error norms have been taken into account :

(a) Integral square error

$$E_2 = \sum_{i=1}^{n} e_i^2 \tag{6}$$

(b) Maximum error

$$E_\infty = \max_{1 \le i \le n} e_i \tag{7}$$

The performance of the algorithm is illustrated in figures 1 to 4.

The algorithm has been implemented in C language on a SUN 3 computer.

5 Concluding Remarks

The proposed algorithm, which can be ascribed to the split and merge class, compares favourably with most other algorithms devoted to corner point detection. Particularly, taking a recent algorithm by Teh and Chin [2] as a measure, we do not only excel in processing time, but in appropriateness of the found corner points as well.

Future efforts should be devoted to improving the hierarchical representation. With the actual procedure, taking the first two iterations provides a coarse, but often sufficient approximation in most cases. A somehow better approximation seems possible if more than one peak is detected in correspondence with certain intervals. In later iterations, we detect often too many peaks and the adopted hierarchical labelling, simply dependent on the size of the supporting arc, is not always easy to be profitably interpreted. Furthermore, for a given substring, we do not detect any additional peak near to the ends of the substring. This is often desirable, but there are cases where some information gets lost. Finally, the triangulation process should be improved in order to avoid the deletion of peaks detected at coarse resolution, which is likely to damage the hierarchical representation.

Acknowledgements

Part of this work has been done during the sojourn of one of the authors (C. Arcelli) at the Department of Computer Science of Shizuoka University. The stay has been fully supported by the Scientific Agreement between the Japan Society for the Promotion of Science and the National Research Council of Italy.

References

[1] L. S. Davis, "Understanding Shape: Angles and Sides", *IEEE Trans. Comp.*, no 3, pp 236-242, 1977

[2] C. Teh and R. T. Chin, "On the Detection of Dominant Points on Digital Curves", *IEEE Trans. Pattern Anal. Machine Intell.*, vol PAMI-11, No 8, pp 859-872, Aug. 1989

[3] G. Gallus and P. W. Neurath, "Improved Computer Chromosome Analysis Incorporating Preprocessing and Boundary Analysis", *Phys. Med. Biol.*, vol 15, no 3, pp 435-445, 1970

[4] H. Freeman, "On the Encoding of Arbitrary Geometric Configurations", *IRE Trans.*, vol EC-10, pp 260-268, 1961

[5] A. R. Dill, M. D. Levine and P. B. Noble, "Multiple Resolution Skeletons", *IEEE Trans. Pattern Anal. Machine Intell.*, vol PAMI-9, No 4, pp 495-503, July 1987

Explanations :

– – – – Input figure

——— Approximated figure

o Dominant or corner-points

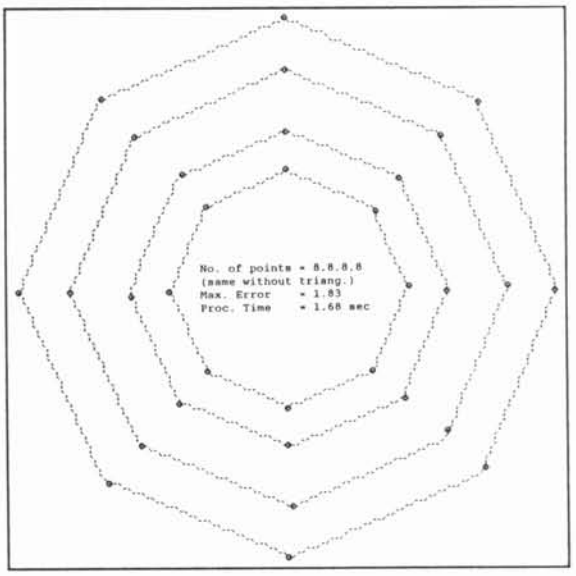


Figure 1: Four octagons of different size. Size invariance is demonstrated for perimeters from 140 pixels up to 320 pixels. Triangulation does not remove any points.

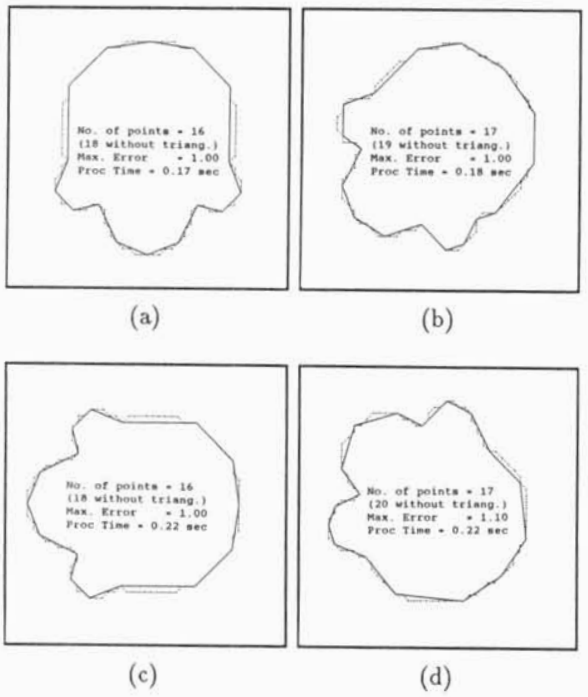


Figure 2: A curve with four semicircles, rotated in memory by 45°, 90°, 135°, respectively.

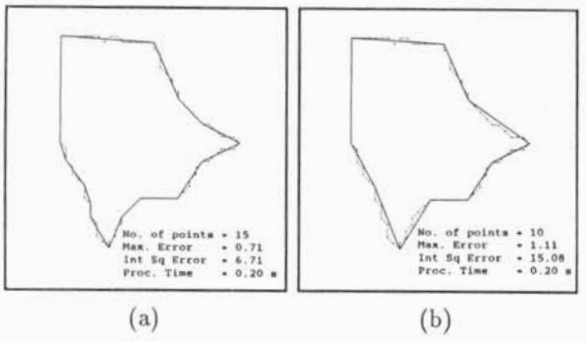


Figure 3: Effect of triangulation on State of Texas. a) Before triangulation. b) After triangulation. Triangulation yields a compression of ca. 1.5.

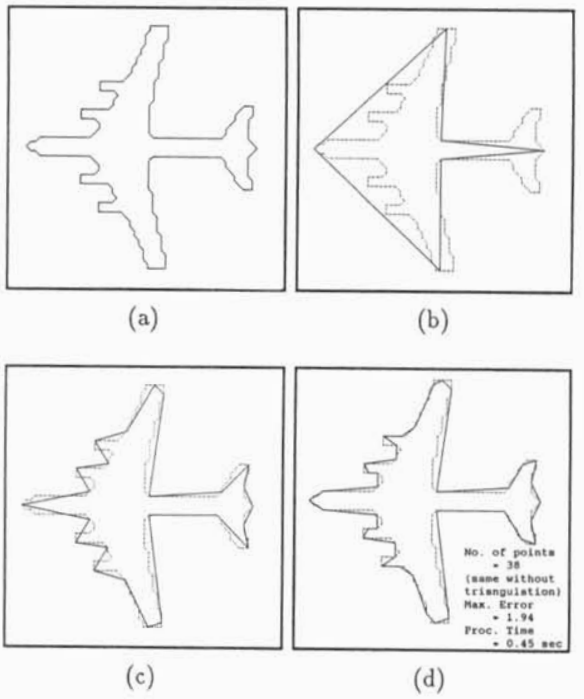


Figure 4: Hierarchical approximation of a plane. a) Input picture. b) Vertices found during first iteration. c) Vertices of first and second iteration. d) Final approximation with three iterations. Triangulation does not remove any points.

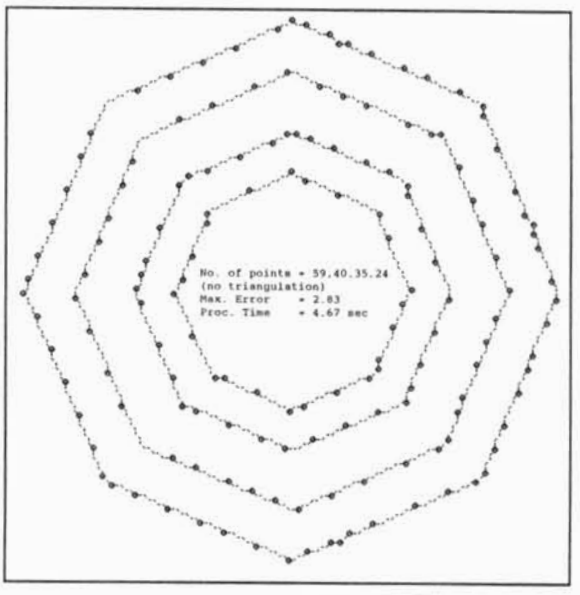


Figure 5: Four octagons of different size. Output for the Teh-Chin algorithm. Please compare with Figure 1.