Corner Detection by Searching Two Class Pattern Substrings

Hermilo Sánchez-Cruz

Centro de Ciencias Básicas. Universidad Autónoma de Aguascalientes
Av. Universidad 940, Col. Universidad, CP. 20100
Aguascalientes, Aguascalientes. México. Fax: (52 449) 9 10 84 01
hsanchez@correo.uaa.mx

Abstract. A new method for corner detection is proposed. Previous approaches for detecting corners rely on computing angle functions to find changes of curvature. Generally, those methods employ eight different symbols to represent contour shapes. The method of this work is based on using three symbols of a chain code to find pattern substrings, detecting corners in the contour shape. The method relies on searching for the relationship among neighbor points, finding two basic pattern contour chain elements, requiring few computing power to obtain shape corners.

Keywords: Corner; Contour; Chain element; Freeman chain code; Three-symbol chain code; Pattern substrings.

1 Introduction

In literature, usually the aim in obtaining corner points by computing angles of curvature on the contours of shapes is studied. Freeman and Davis [1] proposed to find corners by computing incremental curvature to represent contour shapes by an eight-direction chain code. Since then, many authors have suggested to use this code when representing contour shapes. Part of the algorithm presented by Teh and Chin [2] consists on computing the curvature of contour points and detecting corners by a process of nonmaxima suppression. Liu and Srinath [3] have compared a number of corner detectors due to Medioni and Yasumoto [4], Beus and Tiu [5], Rosenfeld and Johnston [6], Rosenfeld and Weska [7] and Cheng and Hsu [8]. All those authors represented samples of shapes through a sequence of eight direction changes from 0-7, known as the Freeman Chain Code [9]. We propose here to use a method of only three relative direction changes (Fig. 1a). Techniques due to Freeman chain codes in finding corner detection are based on eight different directions (see Fig. 1b).

An advantage in using three symbols is its low storage power, as can be seen by the recent work duo to Sánchez-Cruz & Rodríguez-Dagnino [10]. They found that coding with three symbols is sufficient to represent binary shapes saving storage efficiently. However, recently Yong Kui Liu & Boruk Zalik[11], found efficient storage properties in using the eight directions of Freeman chain code.

J.F. Martínez-Trinidad et al. (Eds.): CIARP 2006, LNCS 4225, pp. 354–362, 2006.

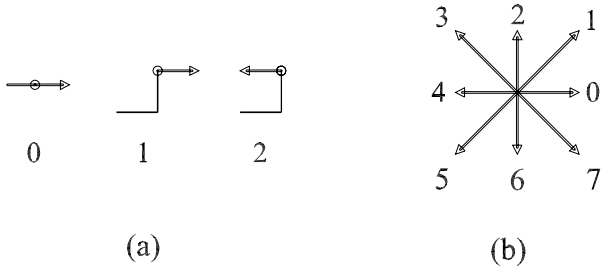


Fig. 1. Two chain codes: (a) Three-symbol, (b) Freeman chain code

However, the number of symbols of Fig. 1a. constitute an advantage in finding a small set of pattern substrings.

For each orthogonal change direction code, chain segments are divided in three parts (given in Fig. 1a.): a *reference segment* (in Fig. 1a. appears as horizontal segment in each code), *a basis segment* (perpendicular to reference segment) and a segment indicating a direction change with regard to reference segment.

The meaning of the three symbols (see Ref[12] for 3D case), given by the set $\mathcal{C} = \{0,1,2\}$, is as follows: the element 0 represents the direction change which means to go straight through the contiguous straight line segments following the direction of the last segment; the 1 indicates a direction change upward with regard to the reference segment; and 2 means to "go back" with regard to the direction of the reference segment. Under certain constrictions, when this particular symbol appears in a contour shape, could easily indicate an existing corner. In Section 2 definitions concerning to this article are presented, seeking the problem as a pattern substring search to obtain corners. In Section 3 some rules to detect corner points are proposed; in Section 4 experimental proving of postulated rules are applied on some binary shapes; and in Section 5 some conclusions are given.

2 Definitions Related to Pattern Chain Substrings

Our proposal method considers to find a specific set of pattern substrings of length l, trying to find all those substrings in a chain code that mach with those patterns. Let us consider, for example $l = 9$ as the length of the substring. Which substrings are all composed of 9 symbols and which of them are considered corner chains? In fact, there are substrings composed of 9 symbols, of course, not all are considered corner chains due to its low curvature or because the region they are associated in the contour shape is not "well behaved", as we explain at once.

Let $\mathcal{P}$ denote the *complete chain code* associated to the shape contour, given by the string of symbols p_i of eq(1).

$$\mathcal{P} = p_1 p_2 \cdots p_n, \tag{1}$$

and P the contour discrete perimeter, given by the number of symbols of the chain code.

Consider a substring template of l symbols: $\mathcal{C} \in \mathcal{P}$, given by eq(2).

$$C = a_1 a_2 \cdots a_l, \quad l \ll P, \tag{2}$$

as a *contour chain element*, or simply: *chain element*, this is, a small piece of contour from the whole shape contour.

Let us consider $m = l/2$ the middle point of a substring of size l, so that a_m, the pivot, be the center of the substring. It is possible to associate a pair of line segments to any chain element. They can be drown up from to the opposite end points, producing an angle φ. We define a *well behaved* chain element when an angle has been subtended by a pair of associated line segments such that the chain does not form loops. In Figure 2 are presented some examples of chain elements. In Figure 2(a), (b) and (d) angles defined by midpoints denoted by a small circle, are well behaved, but that presented by Figure 2(c) is not.

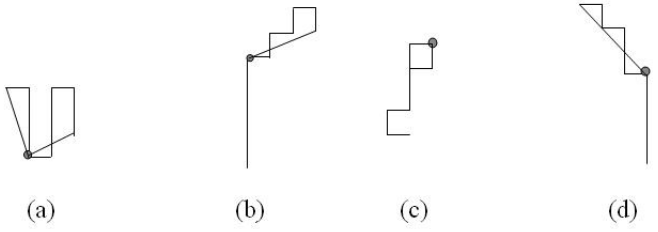


Fig. 2. It is associated a pair of line segments to each of the substring from their middle points

Once well behaved condition is accepted, the angle φ between the two well behaved line segments can be computed, and a threshold is fixed to propose a chain corner.

We define a *neighborhood of radii r*, when considering a piece of the complete string; this region is composed of a small number of symbols in comparing with the whole contour chain code, r symbols on one hand of a particular pivot symbol, and r symbols on the other side of the pivot symbol. An example of neighborhood of radii 5 is: 00000(1+2)10011. This is, a chain element having 00000 in its left first part, 10011 in the right second part, and 1 or 2 (1+2, to abreviate) as a pivot symbol.

To save calculation of corner-angles or curvature changes directly, instead we give a family of substrings and whose angle subtended by well behaved line segments represents a high curvature. Well behaved substrings should not be considered a corner chain when their corner chains associated angles are so much obtuse. Quantitatively, how acute or how obtuse has to be an associated angle to consider a chain element as a corner?

To find pattern substrings, our experimets make us to consider an angle φ associated to a chain corner if $\varphi = [0°, \pm 126°]$ (or $\varphi = [0, \pi \pm \frac{\pi}{3}]$ in radians).

Fig. 3. A well behaved substring and its associated angle

For example, in Fig. 3 a chain element 110110011 is shown, covering the chain from bottom to up, with an associated angle $\varphi = 165.35$, the chain is not considered a corner.

Another definition we need is a *well behaved contour shape*, this is, a contour shape having been smoothed in such a manner that there is no noise or local defects.

3 Rules for Detecting Chain Corners

Part of the study made to find a simple pattern of substrings that represent corners, is to analyse the whole universe of a small vicinity of shape contours, the chain elements. At the begining, to search these patterns, we did our experiments within a vicinity of nine segments in chain elements, giving good results in finding chain corners.

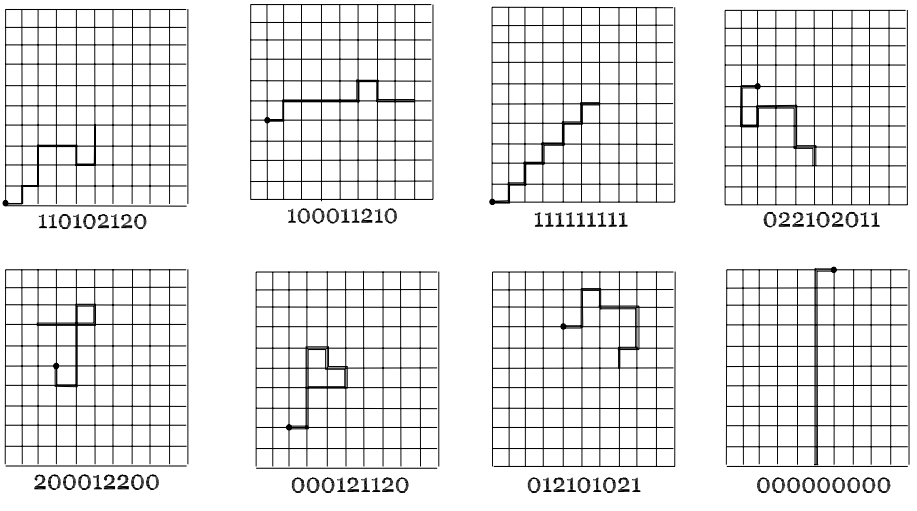


Fig. 4. Eight samples of substrings composed of eleven segments

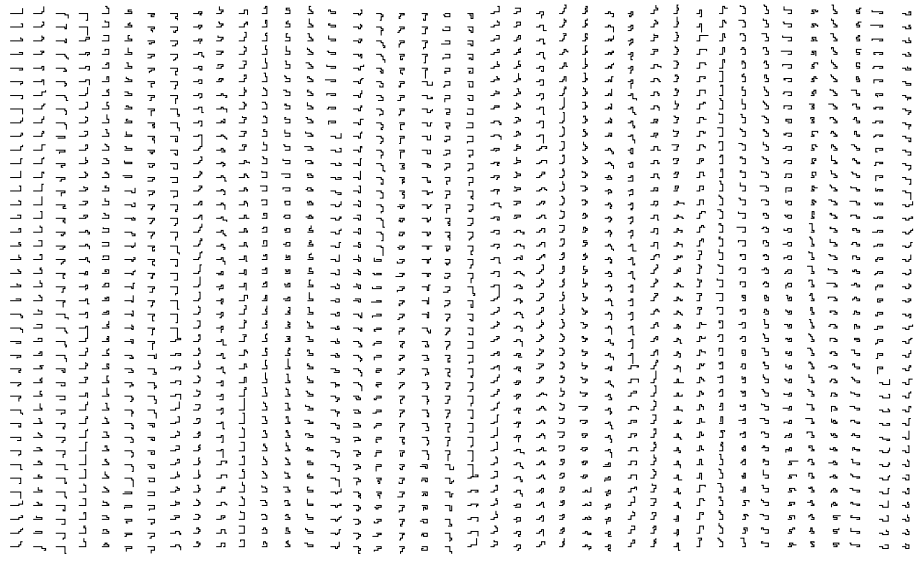


Fig. 5. Part (1 600) of the 6 432 chain elements in the range $[0^o, \pm 126^o]$

We are focused on finding a group of pattern substrings or pattern chain elements considered as chain corners. Eleven chain segments are considered for our study, nine of them are labeled with symbols, representing ortogonal direction changes. The first two are called reference segment and basis segment, respectively. There are a huge number of combinations given by nine symbols (11 segments) in a grid of 10×10. Fig. 4 shows only eight samples.

Even more, fixing the reference segment of the chain, there are 3^9 combinations duo to the other nine chain directions.

There are many possible combinations of chain elements. From this set, we are interested on finding chain elements that have no loops. Looking for these chain elements there are 11 025.

Computing the associated angle to each of these elements, there are 6 432 chain elementes in the range of $[0°, \pm 126°]$. See Fig. 5.

By analizing the different chain sets mentioned, we have observed that pattern substrings representing corners, or even, line elements can be obtained. Parameters we have to take into account are the next:

l: states for the size of substring.

q: represents "many" times a symbol is repeated in a substring. This quantity depends on resolution of binary object. By *many* we define that the number of symbols is greater than the quantity: $l/4$, so $q \in [l/4, l]$.

A way to obtain a complete set of templates considered chains corners, is to search all the substrings arrays composed of l symbols from the set $\mathcal{C} = \{0,1,2\}$, calculate the angle associated to each substring and apply the threshold to see

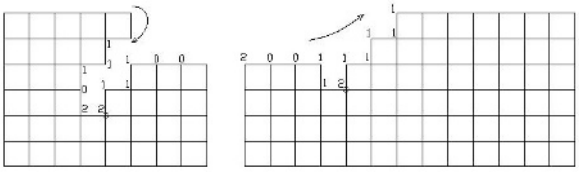


Fig. 6. Examples of chain elements, covering the contour on clockwise sense, of kind S_2. The grid is part of the inner shape.

if it is a chain corner. But we propose a small enough set of template substrings to find the evident chain corners from an arbitrary set of 2D shapes.

As we have computed, from the 6 432 chain elements, there are 4 334 (more than two third parts of the set) chain elements with symbol '2' near the pivot, with associated angles in the range $0° \pm 126°$, whereas only 614 (sligthly more than a third part) from the range $\pm 18°$, that can be considered as a rect lines set (1 547 chain elements). So, to simplify the pattern of chain elements that correspond to chain corners, lets consider only the case where the contour of the image is also a well behaved contour, and consider only those chains that better fit to a pair of associated segments. In this case we are talking about pattern strings of discrete chain corners, postulated by next regular expressions:

$$
\begin{aligned}
S_1 &= (0+1+2)^{l/2}(\mathbf{2})(0+1+2)^{l/2} \\
S_2 &= (0^q+1)^{l/2}(\mathbf{1})(0+1^q)^{l/2} + (0+1^q)^{l/2}(\mathbf{1})(0^q+1)^{l/2} + \\
&\quad (0^q+1)^{l/2}(\mathbf{1})(0^q+1)^{l/2}
\end{aligned}
\tag{3}
$$

where q represents *many* symbols. For example, pattern S_1 means the pivot is a '2' symbol, independently of the symbols on both sides; whereas S_2 means that substring has many zeros or ones behind the middle symbol and many zeros or ones in the second part of the substring, or many zeros on both sides of a one pivot. Our proposed method relies on looking for these pattern substrings on any contour shape.

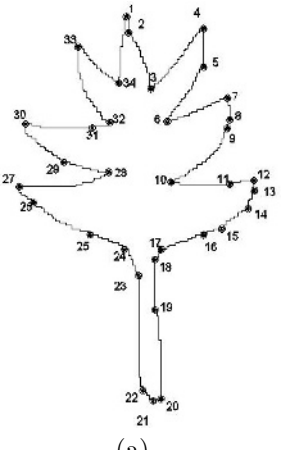
We consider shapes represented by resolution cells, each having a value 0 or 1. Contour shape is covered in the clockwise sense. For the implementation of an algorithm to encode this shape we have to visit the ones that represent the contour shape, i.e., the ones of the boundary. Using the three code symbols we follow the contour of the shape counter clockwise, and we give one of the three relative chain codes according to each orthogonal change direction. A manner to fix every orthogonal change is by defining a 3×3 window, then we choose a starting one as the core of the window, and we analyze its neighborhood by finding directed vectors on the boundary of the shape. Hence, we calculate the changes and produce the code. This procedure continues until visiting all ones of the boundary. The object is confined to a minimum rectangle that is visited line by line, from left to right and from top to bottom. The first cell resolution, of

the object to be visited, is that which appears at the leftmost and highest part of the occupied region. Fig. 6 shows examples of representing part of contour shape by giving the three symbols of the orthogonal directions given in Figure 1a. When we start to go over the contour the first two discrete segments do not represent a direction change with regard to reference segment. When one ends to go over the contour, it is possible to give chain elements at starting point and contiguous direction change because of last reference segments visited. Given this representation, we can reconstruct the original image by interpreting the code of every symbol in terms of the direction changes that can follow. Finally, the pattern substrings, S_1 and S_2 are parsing the resulting chain string of the complete contour.

4 Proving the Postulated Rules

Consider the set of three shapes $\mathcal{S} = \{$*Plant, Hammer, Circles, Tigger* $\}$.

Consider the *Plant* shape object and its corresponding chain code (Fig. 7), 34 chain corners were found in its contour shape. Some of them are so closed, in such a manner that their corresponding pivots are in the neighborhood of each other, in this case we could define only one corner. In Table 1 is listed each of the chain elements and the corresponding class pattern given by eq(3) of the *Plant* shape contour.



(a)

12110000001101101111011001100110011000000121010101111111111011111110111111011111011111101111
10102000000000000000000211101101101101101111110111110111111110110112021001100110011100110011011011
01101101102000110000002100011011001100110111111111111111011010101101111110011011012000110000000000
00000000021000011000000200002100011110001110110101011011010111110001100110110001101100110011100111
10010110111100000000000000000000000021001100000000000011000000000000000000000000000000211002101011111101
10000110000000000000000000000000000000000000000000000000000011011001100110110010111101100110110011000 1
11100011000101001100110111111011010111001011111011110111002000000000000000000000000011001100001111 0
01101101200001100011000110011001100111011111011111110111101101111000020021000110000000000000000 0
00000002100000000102011010110101011110111111001111000011001100000000000202110111101111111101011110111
11011111120011000000000000001100000110000000

(b)

Fig. 7. *Plant* shape and its corresponding 3-symbol chain code: (a) the shape; (b) its chain code

Table 1. Chain elements encountered from the Fig. 7 that belong to one of the two classes of chain patterns postulated. Corners 6,30 and 33 are split by two closed chain corners.

Num corner	*Chain element*	*Class pattern*	*Num corner*	*Chain element*	*Class pattern*
1	00001211000	S_1	19	00000210011	S_1
2	00000110110	S_2	20	00000211002	S_1
3	00001210101	S_1	21	21100210101	S_1
4	11010200000	S_1	22	00110110010	S_2
5	00000211101	S_1	23	00110110001	S_2
6a	11011202100	S_1	24	10001111000	S_2
6b	01120210011	S_1	25	11000101001	S_2
7	10110200011	S_1	26	01001100110	S_2
8	00000210001	S_1	27	11100200000	S_1
9	10001101100	S_2	28	01101200001	S_1
10	01101200011	S_1	29	10001100110	S_2
11	00000210000	S_1	30a	10000200210	S_1
12	00000200002	S_1	30b	00200210001	S_1
13	00000200002	S_1	31	00000210000	S_1
14	10001111000	S_2	32	00010201101	S_2
15	00011110001	S_2	33a	00000202110	S_1
16	10001100110	S_2	33b	00020211011	S_1
17	00110110001	S_2	34	11111200110	S_1
18	10001101100	S_2			

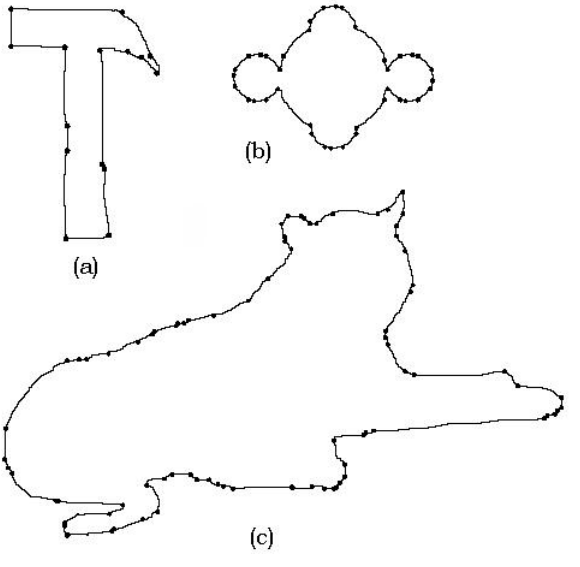


Fig. 8. Sample shapes, (a) *Hammer*, (b) *Circles* and (b) *Tigger* shapes and their corresponding corner points

Figure 8 shows the results of applying the method proposed to search chain corners on the sample objects.

5 Conclusions

With this method we have found shape corners including where shape is apparently circular, like happening with figure constructed by intersecting circles. We used three symbols to represent binary shapes, implying to save time and memory storage to manage this kind of objects. We found patter substrings to obtain most important shape corners in contour shapes preventing to compute angles and curvatures. The method, also, is suitable to find corners on regular shapes, like *Hammer*, or on irregular shapes, like *Tigger*. We have presented a new research topic, in avoiding computing explicitly angles and curvatures. A universal and simplified set of pattern substrings, comparing with other chain codes in literature is suggested to be investigated. As future work most be studied if this method is invariant under scale and rotation transforms.

Acknowledgments

We would like to thank PROMEP program and CONACyT council for their support in finishing this work.

References

1. Freeman, H. and Davis, L. S.: A Corner-Finding Algorithm for Chain-Coded Curves. IEEE Trans. Comput. 26: (1977) 297-303.
2. Teh, C-H. and Chin, R.T.: On the Detection of Dominant Points on Digital Curves. IEEE Trans of Pattern Anal and Mach Int. 11 (8) (1989) 859-872.
3. Liu, H-C; Srinath, M.D.: Corner Detection From Chain-code. Pattern Recognition. 23 (1/2) (1990) 51-68.
4. Medioni, G.; Yasumoto, Y.: Corner detection and curve representation using cubic B-Splines. Comput. Vision Graphics Image Process. 39: (1987) 267-278.
5. Beus, H.L.; Tiu, S.S. H.: An improved corner detection algorithm based on chain-coded plane curves. Pattern Recognition. 20 (1987) 291-296.
6. Rosenfeld, A.; Johnston, E.: Angle detection on digital curves. IEEE Trans Comput. 22: (1973) 875-878.
7. Rosenfeld, A.; Weszka, J.S: An improved method of angle detection on diginal curves. IEEE Trans. Comput. 24: (1975) 940-941.
8. Cheng, F.; Hsu, W.: Parallel algorithm for corner finding on digital curves. Pattern Recognition Lett. 8: (1988) 47-53.
9. Freeman, H.: On the Encoding of Arbitrary Geometric Configurations, IRE Trans. on Electr. Comp. 10 (2) (1961) 260-268.
10. Sánchez-Cruz, H.; Rodríguez-Dagnino, R. M.: Compressing bi-level images by means of a 3-bit chain code. Optical Engineering. SPIE. 44 (9) (2005) pp 1-8. 097004.
11. Liu, Yong K.; Zalik, B.: An efficient chain code with Huffman coding. Pattern Recognition 38 (4) (2005) 553-557.
12. Bribiesca, E.: A chain code for representing 3D curves. Pattern Recognition. 33(5)(2000),755-765.