Polygonal Approximation of Contour Shapes Using Corner Detectors

Hermilo Sánchez-Cruz*[1]; Ernesto Bribiesca[2]

[1]Departamento de Ciencias de la Computación. Centro de Ciencias Básicas. Universidad Autónoma de Aguascalientes, Aguascalientes,México.
* hsanchez@correo.uaa.mx
[2]Departamento de Ciencias de la Computación. Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas. Universidad Nacional Autónoma de México. D.F.,México.

ABSTRACT
A great amount of corner detectors that appear in literature are based on using the Freeman chain code of eight directions, which is used to represent contour shapes. We propose a new method for corner detection based on a three-symbol chain code representation, which requires lower storage memory and an easy way to obtain shape corners. We compare it with five existing methods, which are well known in the literature, giving our method a better performance. Furthermore, in order to reconstruct the original shapes through polygonal approximations, we propose an error parameter to quantify the efficiency. This can be accomplished by considering the redundancy of points produced when looking for corners and when computing the difference between the original region and the approximated polygon.

Keywords: Binary objects, corner detection, pattern strings, polygonal approximation, 3OT chain code.

RESUMEN
Gran parte de los detectores de esquinas, que aparecen en la literatura, están basados en el uso del código de cadena de Freeman de ocho símbolos, el cual es usado para representar los contornos de las formas. En este trabajo presentamos un nuevo método para detectar esquinas basado en una representación de código de cadena de únicamente tres símbolos, lo que requiere menor consumo de almacenamiento en memoria y permite, de manera sencilla, obtener esquinas en los contornos de las formas de los objetos. Dicho método lo comparamos con cinco métodos altamente citados en la literatura, dando nuestro método un mejor desempeño. Más aún, con el propósito de reconstruir las formas originales a través de aproximaciones poligonales, hemos propuesto un parámetro de error para cuantificar la eficiencia de cada detector, el cual se logra al analizar la redundancia de puntos que aparecen al tratar de localizar las esquinas, así como de establecer una diferencia entre la región original de la forma y la debida a la aproximación poligonal.

Palabras clave: Objetos binarios, detección de esquinas, patrones de cadenas, aproximación poligonal, código 3OT.

1. Introduction

Attneave [1] found that some characteristic points are determinant in shape recognition. According to Atteave's work, when the mentioned points are joined by straight lines, the resulting shape should be very similar to the original. Our present work is inspired by this observation: In closed curves, we try to find as minimum characteristic points as possible and sufficient to obtain approximately the original shapes. Also, we take advantage of the compression properties of a chain code composed of three symbols. One representing no change directions, and each of the other two indicating orthogonal changes of direction [2].

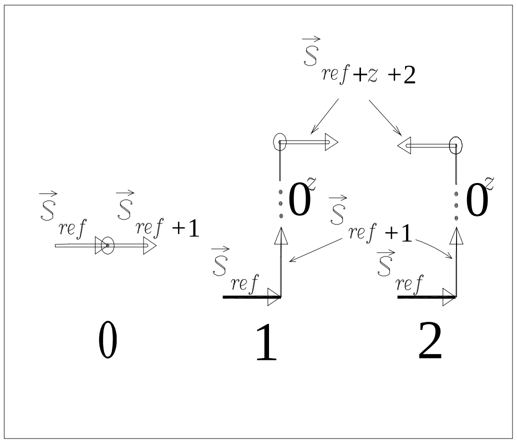
The corner detection of the shape of objects is an active field in object recognition and image retrieval. In literature, usually, to obtain corner points, computing angles of curvature on the contours of shapes are carried out and, to represent discrete contours, Freeman chain codes are generally used. Freeman and Davis [3] proposed to find corners by computing incremental curvature in contour shapes represented by an eight-direction chain code. Since then, many authors have suggested to use this code to represent contour shapes and to look for corner points. Part of the algorithm presented by Teh and Chin [4] consists of computing the curvature of contour points and detecting corners by a process of nonmaxima suppression. Liu and Srinath [5] compared a number of corner detectors due to Medioni and Yasumoto [6], Beus and Tiu [7], Rosenfeld and Johnston [8], Rosenfeld and Weska [9], and Cheng and Hsu [10]. All those authors represented samples of shapes through a sequence of eight direction changes from 0-7, known as the Freeman Chain Code [11]. Techniques due to the Freeman chain codes in finding corner detection are based on eight different directions (see Fig. 1b).

Wu [12] proposed an adaptive method to find local maximum curvatures of digital curves. Sobania and Evans [13] described a corner detector from segmented areas using the mathematical morphology and employing paired triangular structuring elements. A disadvantage of their work is that it can be slow due to its high computational complexity. Basak and Mahata [14] developed a connectionist model along with its dynamic states for detecting corners in binary and gray level images.

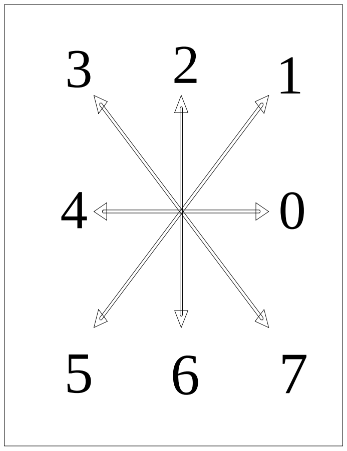
Previous approaches to detect corners using chain codes are given in literature. By measuring the number of links to both sides of a point that can produce the largest digital straight line, Koplowitz and Plante [15] proposed a corner detection scheme for Freeman chain coded curves. Subri et al. [16] presented a neural network classifier, by using the Freeman chain code, obtaining corners on the boundary of a sample line drawing. Arrebola and Sandoval [17] proposed a method to characterize a curve by means of the hierarchical computation of a multi-resolution linking approach. They adapted the multi-resolution linking algorithm to the processing curve contours described by the Freeman chain code. Also, Marji and Siy [18] developed an algorithm to detect corner points on an 8-connected shape by using the Freeman chain code in a polygonal approximation.

In [2], the authors proposed that it is suitable to represent any binary closed shape with binary resolution cells by means of the use of only three symbols of a chain code without loss of information. This method of chain code is sufficient to represent binary shapes and represents a low cost alternative in terms of storage memory.

The chain code used in our work consists of the set 3OT = {0,1,2} and is given in Fig. 1(a) (see [19] for the 3D case).



(a)



(b)

Figure 1. Two different chain codes: (a) The three symbols used to indicate a possible change direction, (b) the eight different directions given by the Freeman chain code.

A simple closed contour shape can be represented by a discrete closed curve given by the sequence:

$$C = \{p_i = (x_i, y_i) \mid i = 1,2,\ldots n\} \quad (1)$$

where n is an integer and the initial point p1 is an immediate neighbor of the final point

pn of the curve.

When covering the discrete curve, two consecutive points, pi,p(i+1)modn, conform a unit vector in one of the x,y directions of the Cartesian Plane, i.e.

$$\vec{s}_i = (p_i, p_{(i+1)\mathrm{mod}n}) \quad (2)$$

and have the directions of the unit vectors { along the x,y axes in the Cartesian Plane,

$$\vec{s}_i = \{-\hat{\iota}, -\hat{j}, \hat{\iota}, \hat{j}\} \quad (3)$$

Let us take any vector as a reference vector in one of the multiple steps needed when traversing the discrete contour curve: $\vec{s}_{ref} \equiv \vec{s}_i$ Thus, when two consecutive vectors have the same direction, we label the point of the curve with a symbol 0, when a change of direction occurs, regarding the reference vector, we put a symbol1 or 2, as follows:

$$\begin{array}{ll} \text{symbol 0 when } \vec{s}_{ref} = \vec{s}_{ref+1} & \\ \text{symbol 1 when } \vec{s}_{ref} = \vec{s}_{ref+z+2} & \text{if } \vec{s}_{ref} \cdot \vec{s}_{ref+1} = 0, \quad (4) \\ \text{symbol 2 when } \vec{s}_{ref} = -\vec{s}_{ref+z+2} & \text{if } \vec{s}_{ref} \cdot \vec{s}_{ref+1} = 0, \end{array}$$

where *z* is a zero or positive integer, and its value depends on the number of unit steps given in one same direction. The symbol 0 is represented by two vectors: a *reference vector* and a vector indicating a direction change. In this case, the direction change is the same as that of the reference vector. For the other two orthogonal direction changes, chain vectors are divided in three parts: a reference vector (in Fig. 1(a) appears as a horizontal vector in each code), a *support vector* (a vector depending on *z*) perpendicular to the reference vector, and a vector indicating a direction change with regard to the reference vector.

So, theelement 0, in the 3OT code, 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 forward change of direction with regard to the reference vector; and ‘2’ means to “go back” with regard to the direction of the reference vector.

Recently, Sánchez-Cruz [20] proposed a new method to find corners by using pattern strings. He found three classes of strings to avoid computing angles and maximum curvatures in contour shapes. His method is based on the three relative direction changes of the 3OT code. Pattern strings given in [20] contain certain constraints and find corners at certain approximations. However, with such a method, a high redundancy in the number of corners can be produced, i.e., a lot of corner points can appear in regions where no high curvature is presented. Now, in this work, we propose another method, apply it to more shapes and compare it with highly cited methods.

An advantage of using three symbols is its low storage power, as can be seen by the recent work due to Sánchez-Cruz and Rodríguez-Dagnino [2]. They found that coding with three symbols is sufficient to represent binary shapes saving storage efficiently.

In Section 2, definitions concerning to this article are presented, seeking the problem as a search of pattern substrings. In Section 3, some rules to detect shape corners are proposed; in Section 4, experimental proving of postulated rules are applied on some binary shapes, and an error given by polygonal approximations to original shapes is computed. In Section 5, we give some conclusions.

2. The chain coded discrete contour

Let us associate one of the three symbols s_i given in Eq.(4) to each vector $\vec{s}_i$: 0,1, or 2. In this work, a specific set of pattern substrings of length *l* is applied to find all those substrings in a contour shape, coded by a chain that matches the set. Which substrings are all composed of *l* symbols and which of them are considered corner chains? Not all substrings are considered corner chains due to their associated low curvature.

Let S denote the *complete contour string* (that represents the *chain code of the shape* associated to the shape contour, given by the string ofsymbols s_i of Eq.(5)).

$$S = s_1 s_2 \dots s_n \equiv [s_i], \tag{5}$$

where *n* is the contour discrete perimeter, given by the number of symbols of the chain code.

Consider a substring $C_j \in$ S given by Eq.(6).

$$C_j = s_j s_{(j+1)\bmod n} s_{(j+2)\bmod n} \dots s_{(j+l-1)\bmod n}, \tag{6}$$

so that such a small contour string of symbols *l* is considered an *elemental contour substring* (*stringel,* for short), i.e., a small piece of contour representation from the whole shape contour that has length *l,* so that *l* << S, *j* states the *j*-th substring in the complete contour string.

Let *m* = (*j*+l) / 2 be the middle point of a substring of size *l* so that the middle symbol, $s_{m(modn)}$, is the center of the substring. It is possible to associate a pair of line segments with any symbol. A *well behaved stringel* is defined as a substring which is associated to a pair of line segments so that the chain does not form loops. Observe a sample of stringel and their corresponding visual meaning in Fig. 2.

We define a *neighborhood of radii r* when a piece of the complete string is considered; this region is composed of a small number of symbols in comparison with the whole contour chain code, *r* symbols on each side of a particular middle symbol:

$$N(s_m, r) = \{s_i \in S \mid |s_i - s_m| < r\} . \tag{7}$$

Fig. 2 presents an example of a neighborhood of radii 5: stringels having 00000 in their left first part, 11110 in the right part, and s_m = 2 as a middle symbol could appear on this piece of contour shape.

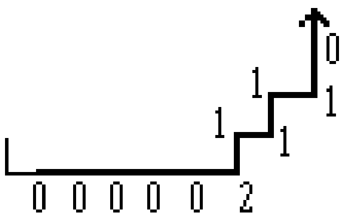


Figure 2. Stringel of 5-neighborhood of middle symbol 2: 00000211110.

3. Pattern strings

A subset of stringels that we call *stringpatts* to find chain corners from an arbitrary set of 2D shapes is given in this section. Stringpatts are patterns of elemental contour substrings similar to those found in [20] (of course, polygonal approximation was not dealt with in such a paper), but we now modify them by releasing some constraints to symbol '1' in S_2 and S_3. To find a group of stringpatts considered as chain corners, we focus on a vicinity of each change code contour, for example of radii eleven, i.e., eleven chain directed segments, labeled with symbols and representing direction changes.

So, in order to simplify the stringpatts that correspond to chain corners, we postulate the following regular expressions:

$$\begin{aligned} S_1 &= (0+1+2)^{l/2}2(0+1+2)^{l/2}, \\ S_2 &= (0^q+1)^{l/2}1(0^q+1)^{l/2}, \\ S_3 &= (0^q+1)^{l/2}1(0+1^q)^{l/2} + \\ &\quad (0+1^q)^{l/2}1(0^q+1)^{l/2}, \end{aligned} \tag{8}$$

where $q > l/4$ represents *many* symbols. The first stringpatt pattern is composed of a string of any symbols, then a symbol '2' and then a sequence of any symbols again.

Another stringpatt that could represent a change direction occurs when there are many 0s (with possibly some 1s) following a '1' as a middle symbol of the stringel, followed again by many 0s (with possibly some 1s).

A third stringpatt is found when there is a substring of many 0s, with possibly some 1s, following a substring of many 1s (with possibly some 0s); or vice versa, a substring of many 1s (with possibly some 0s) following a substring of many 0s (with possibly some 1s), having the 1 as a middle symbol.

Fig. 3 shows the three ideal cases of regions of support, given by the rules of Eq.(8). Of course, not all the stringpatts of Eq. 8necessarily constitute real corners, but they are candidates. See Fig. 4. There could appear stringpatts that satisfy Eq. (8), however, not all of them are corner points.

Our proposed method relies on looking for these stringpatts on any contour shape, without the paired constrictions in Eq.(8) of Ref[20] and adding the symbol '2' in S_1 of Eq.(8). Instead, we introduce other constrictions: we group dots as candidates to be corners given a certain threshold in a vicinity distance.

On the other hand, we consider shapes represented by resolution cells, each of them having a numerical value equal to zero or one. The contour shape is traveled through clockwise. 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 give one of the three relative chain codes according to each change direction.

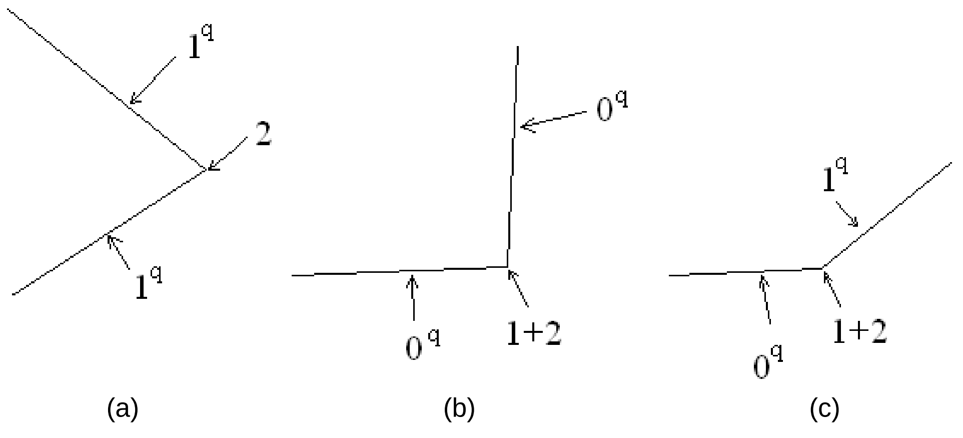


Figure 3. Samples of the three types of corners, each invariant under rotation transformation.

A manner to fix every symbol s_i, that represents a change in direction is by defining a 3 × 3 window, then we choose a starting one as the center of the window, and we analyse 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 all "ones" of the boundary are visited.

The contour of the object shape 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. 4 shows part of a contour shape. When starting to follow the contour, the first two discrete segments do not represent a direction change with regard to the reference segment. When one ends to traverse the contour, it is possible to give a direction change at the starting and contiguous points because of the last reference segment 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, S_2 and S_3 are used to parse the resulting chain string of the complete contour.

3.1 Algorithm

Let $s_m \equiv piv$ be the middle symbol of a stringel. If middle symbols P = $\{piv_b,\ piv_{b+1}\ \ldots\ piv_{b+k}\}$ are separated by a distance smaller than certain value ν, obtainan average of the middle symbol points and define a corner. If middle symbols are separated by a distance greater than ν, consider them as corners. The algorithm of the method is as follows:

```
1. Obtain [$s_i$].  /* The complete contour shape is represented by a string */
2. Obtain $C_r \in S_i$.  /* $C_r$ are the stringpatts */
3. Find a subset $D \subseteq C_r$  /* D is the set of stringpatts representing corners */
executing the following:
for j = 1...num_pivots do
    if $d(piv_j, piv_{(j+1)modnum\_pivots}) \geq \nu$ then $piv_j \in D$
```

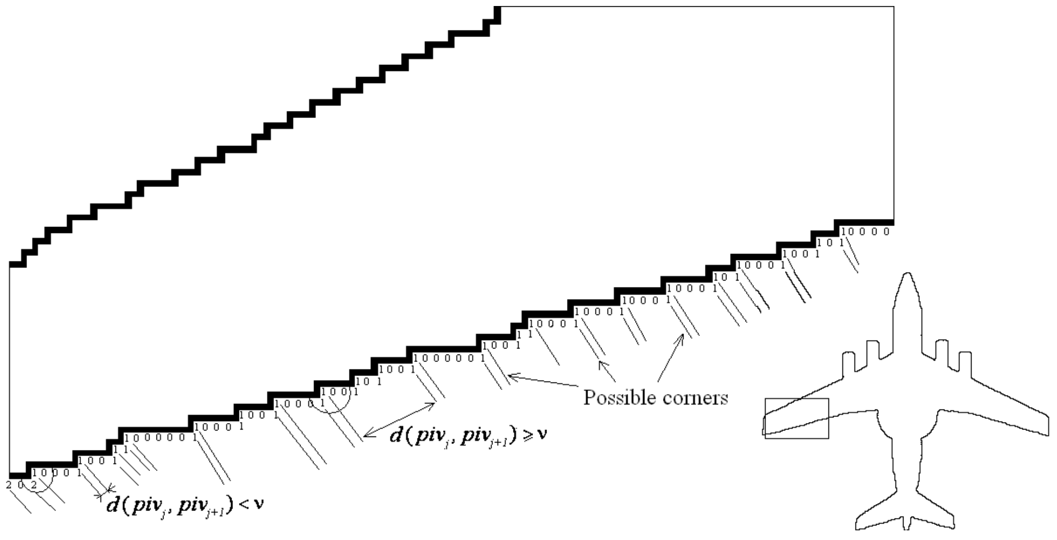


Figure 4. Examples of stringels covering the contourclockwise.

```
else if d($piv_j$,$piv_{(j+1)}$) < $\nu$ /* then analyse whether these pivots are part of a cluster of neighbor pivots */

init = j;

while d($piv_j$,$piv_{j+1}$) < $\nu$
{
temp [i] = j;
j++;
i++;
} /*end while*/

if i < s /* s is a positive integer near 1.  */
p = Average(temp[i])
$piv_p \in D$;
else
$piv_{init} \in D$;
$piv_i \in D$;

/* end for */
```

4. Comparing with other methods

Since the digital curves are represented by discrete contours, the most of the methods do not avoid redundancy, including when apparently straight lines in shapes are observed.

We compare our method of searching substrings with the most frequent corner detectors appearing in literature: Rosenfeld and Johnston (RJ73) [8]; Rosenfeld and Weska(RW75) [9]; Freeman and Davis (FD77) [3]; Beus and Tiu (BT87) [7] and Chetverikov and Szabo (IPAN99) [21]. We call our method 3OT, because it is based on the 3OT chain code to represent discrete contour shapes.

To find corner points, there are some drawbacks in the existing methods: one is that the calculated region of support avoids obtaining angles of small curvature, even when the shape is represented by means of the Freeman code: successive slope angles on the discrete curve can differ by multiples of 45º. So, when modifying parameters of the existing corner detector methods, redundancy appears by finding corners even in apparent straight lines of the shape regions.

All these methods find corner points by using input parameters. While modifying these parameters, a number of corner points appear in the contours. Fig. 5 shows the sample of contour objects frequently used to probe corner detectors.

Considering the object (h) (the Plane of Fig. 5) and its corresponding chain code (Fig. 6), 286 stringpatts were found in its contour shape. Many of them are so closed, in such a manner that their corresponding middle symbols are in the neighborhood of each other, or they are part of cluster points, using the algorithm of Section 3. A subset of these strings correspond to corner chains.

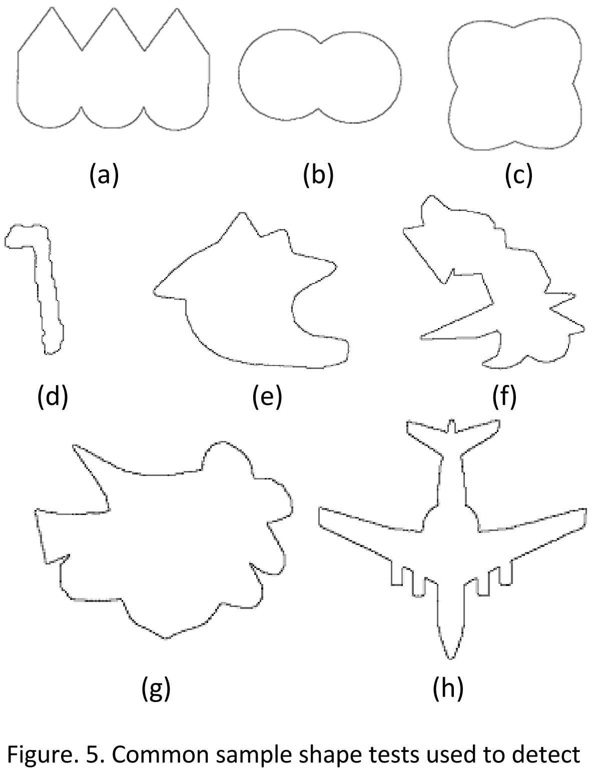


Figure. 5. Common sample shape tests used to detect corners; (a) *Peaks*; (b) *Circles*; (c) *Bubble*; (d) *Stick*; (e) *Spot*; (f) *Irreg*; (g) *Pige*; (h) *Plane*

```
2110110011001111000110011001111000110000011000011011000000000000000000000000000021210000000
0000000000011111011011011011011110201100000000000000000011101100021210210100000000000000000
0000000000101111110011011002110000000000000000000111000000000211000000000000000000000000001
1101111011011011011110011011110011011110011001111011011001101111001101101101101101101110110
1111001101101100111100110110111111100000000000000000020210001100111100000110001100110000110
0110110011000001100111100011000110001100011011000110011011000011001100000110000000000110000
1100021021000110000021111000110000001100111101101111011111010101101101110021212121210000000
0000212100210011000000000000110000000011000011011000211101011011110111111011011111101101111
0110111101111011111101101111100000000110021121011000001100001111212101100110000111100000110
0011000111100211000000110110001011211001100000000001000211100001100011001111000110000011001
1000110001100111100000110020000000210001111111101111010100111101111011111011011011110111100
1111110111111011111111000021000000011000000110110000000000000001100000000000000000000001110110
1111111011111111001101100110001102121100021100210000000001100000000110000000001100000011000
1111000011000110011000011000011001101100011001101100001100011000110001100011001101100000110
0001100001100200002100000000000011011010110110110110001101101101100110111111001101111000110
0101001101101101100111101100110110110110110011110011110011011011001110110210000000000000000
0000000211011021210002111100000000000000000000100021011011110110111000000000000000000000000
0100000000021111000000000000000000111002101101101111011110100000000000000000000000000000000
000000000000011011000000110000110011011000110011011001100110011101100
```

Figure 6. 3OT Plane's chain code with length P = 1 689.

4.1 Polygonal approximation

Fig. 7 shows corner detections using the most known detectors including the proposed one in this work.

Let us present a comparison of our method with that of Rosenfeld & Johnston (RJ73) [8] when looking corner points through the κ parameter.

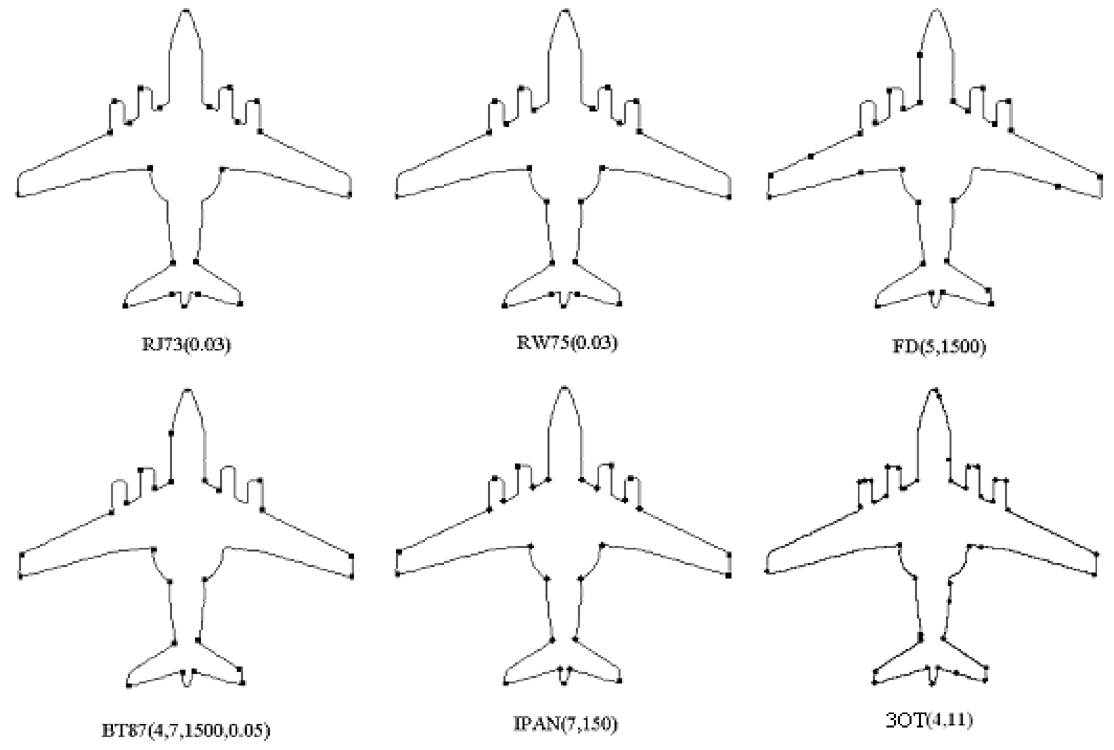


Figure 7. Corner points calculated with all methods at standard parameter values.

Given the RJ73 algorithm, we calculate corner points with different values of its κ parameter. The cases $\kappa > 0.006$ do not reconstruct the contour shape (there are missing points), as can be seen in Fig. 8. Observe in Fig. 9 that the *Plane* shape is well represented by corner points when $\kappa = 0.007$.

On the other hand, in Fig. 10 we show corner points calculated with the RJ73 method at different *k* parameter values. Observe that whereas the parameter value is modified, different number of dots is obtained, and they can help us to reconstruct the shape by joining corner points, and so, fitting a polygon.

Generally, the methods try to employ few corner points to represent the object. However, in this work we consider the necessary corner points to reconstruct the original shape, or an approximation of it. We could say that a *method tends to be good* when it reconstructs the original

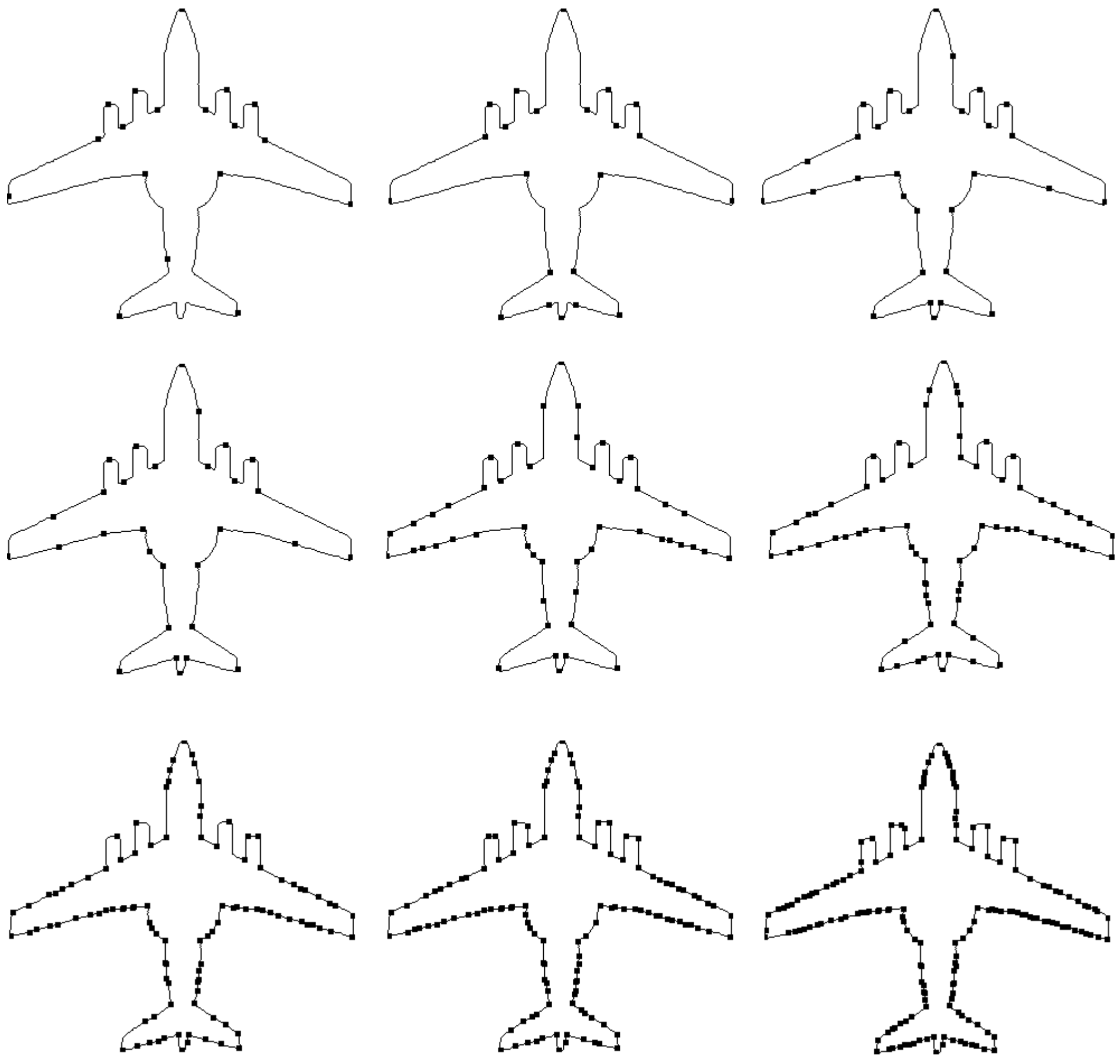


Fig. 8. Corner points calculated using the RJ73 method at different *k* parameter values. From left to right and up to bottom: (a)0.05, (b)0.03, (c)0.02, (d)0.015, (e)0.013, (f)0.01, (g)0.008, (h)0.007, (i)0.006

shape in such a way that the error produced between the original shape and its polygonal approximation is as small as possible.

We have computed the corner points with the different methods. See Fig. 9 where the parameter values of each method appear in parentheses. These parameter values are given by default in the web page [21].

Our method tries to find corner points with less redundancy than the other methods. This is, there are less corner points in straight lines, and also, follow with better approximation the contour shapes.

Observe that our method is better than existing ones to obtain as few as possible corner points in order to reconstruct the original shapes.

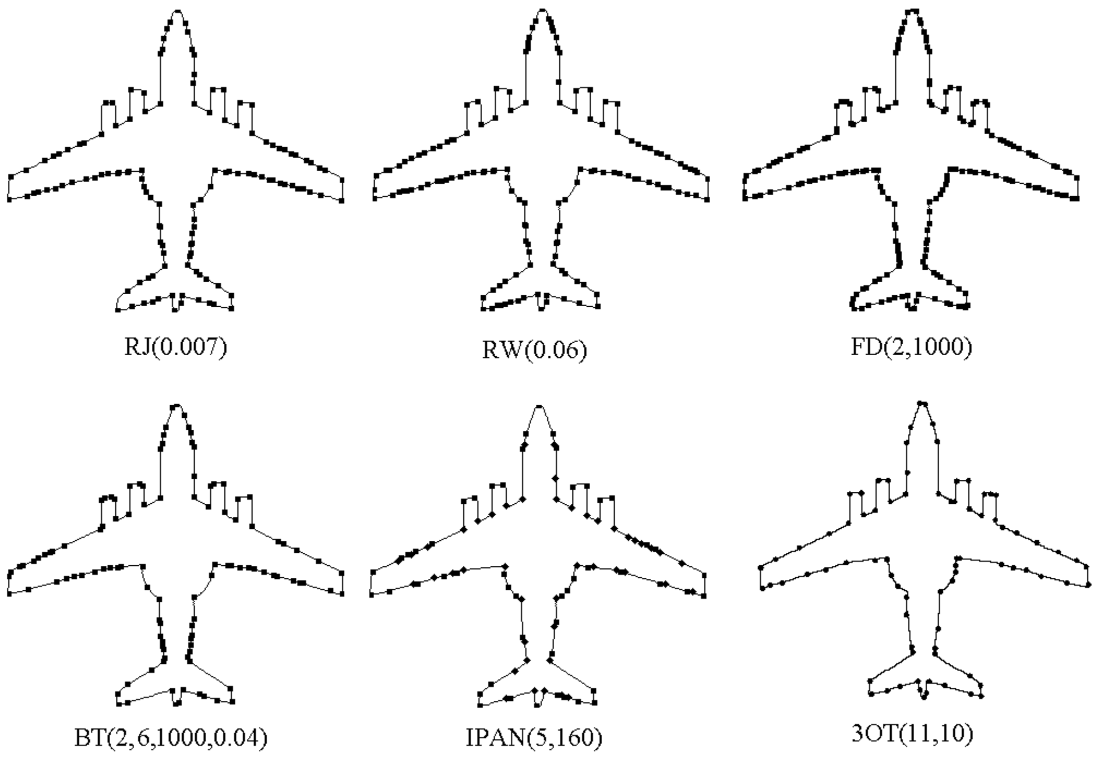


Figure 9. Comparison of the different methods to obtain corners applied to Plane

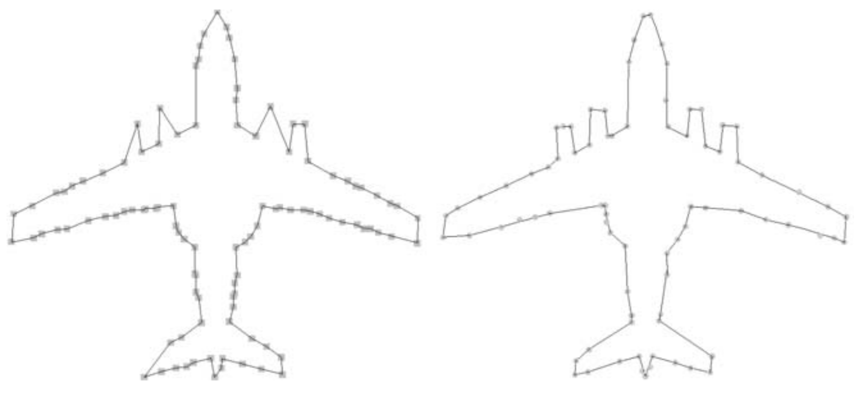


Figure 10. Comparison of the RJ73 (with κ=0.008) and 3OT (*l*=11 and ν=10) to reconstruct Plane object

In Fig. 11 and 12 the reconstructions are shown by a polygonal approximation of the Pige and Spot shape, respectively, using the different algorithms.

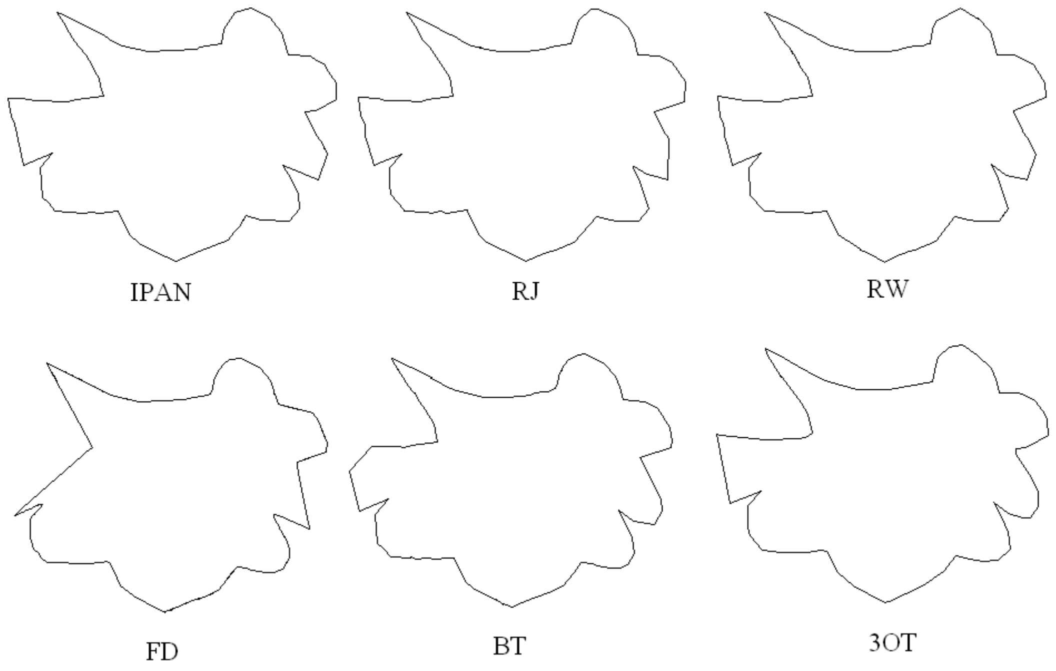


Figure 11. Reconstruction of Pige shape using the different methods.

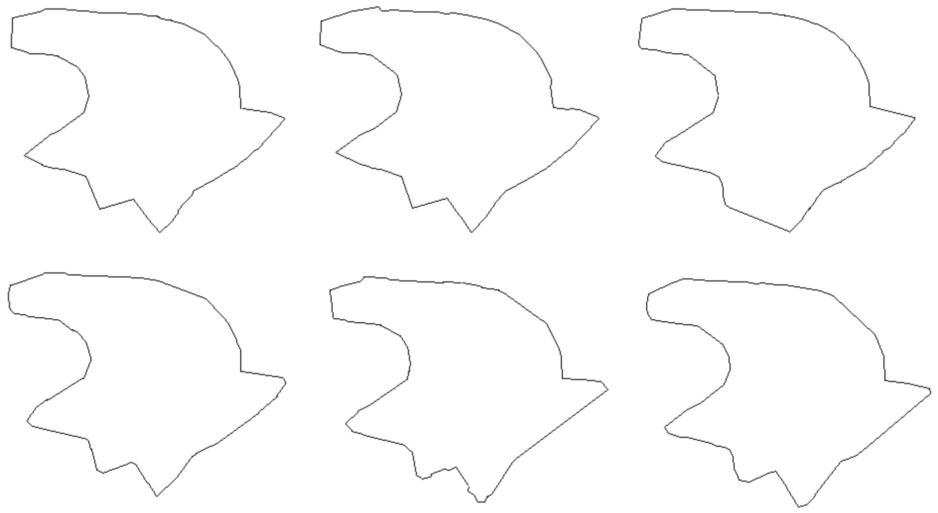


Fig. 12. Reconstruction of Fig. 6(e) (Spot) using the different methods, from the upper left corner to the down right corner: IPAN, RJ, RW, FD, BT, 3OT, respectively.

4.2 Comparing original shapes with polygonal approximations

To test our method, we have compared the fitted polygons by applying the different methods to the original shapes, and we introduced a parameter that measures the error between the polygonal approximation and the original contour.

Let us suppose a given contour shape (Fig. 13(a)), and another given by its polygonal approximation after applying a corner detector (Fig. 13 (b)). For each corner method a polygonal approximation is obtained (Fig. 13(c)); and an error in area, ΔA, is produced. Fig. 13(d) illustrates the error area of the shapes, where C^+ is the set of, say, positive pixels, belonging only to the original shape, and C^- the set of, say, negative pixels, belonging only to the polygonal shape.

The error obtained by the polygonal approximation contains the ΔA area not superimposed in the intersection of both shapes, i.e., $(C^+ + C^-)$, normalized by the area of the original shape, and weighted by the number of corners found by a given method; of course, the smaller the number of corner points to reconstruct the original shape, the better the corner detection. Eq. (9) illustrates this error.

$$\varepsilon = \frac{\text{num\_corners} * (C^+ + C^-)}{\text{OriginalArea}} \tag{9}$$

To compute this parameter, we have obtained the original area of our shapes, which are given in Table 1.

Table 2 gives the next quantities: the number of corners found to reconstruct each contour object, its resulted area of polygon, the number of pixels in the intersection between the resulted polygon and original shape, the non common pixels, and the error parameter. Also, the average error of all methods for each shape is given. In this case, the average error for each test sample suggests that all methods tend to be less efficient for shapes with so many corners than those of smoother contours.

Table 3 shows the global average error committed by each method. As can be noted, 3OT has the smallest average error obtained to reconstruct contour shapes.

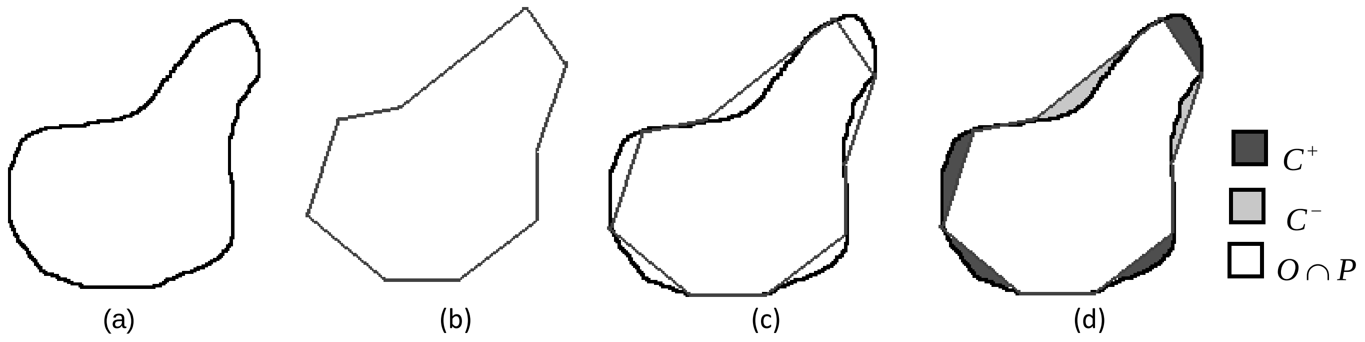


Figure 13. Polygonal approximation of a shape: (a) original shape *O*; (b) its approximation by a polygon; (c) linked corners by straight lines in the original shape, (d) positive, negative and common pixels, respectively.

Shape	*Original Area*(*pixels*)
Peaks	48703
Spot	33869
Irreg	28446
Pige	40703
Plane	20508
Circles	33843
Stick	5344
Bubble	42818

Table 1. Original area, given in number of pixels, of the original contour shapes

Contour Object	*Method*	*num_corners*	*Polygon area*(pixels)	$O \cap P$	*C+*	*C-*	***e***
Peaks	IPAN	53	49924	48683	20	1241	1.3722
	RJ	52	49496	48593	110	903	1.0815
	RW	49	49536	48626	77	910	0.9930
	FD	59	49836	48657	46	1179	1.4839
	BT	49	49316	48396	307	920	1.2344
	3OT	48	48886	48305	398	581	0.9648
Average							1.1883
Spot	IPAN	68	34317	33604	265	713	1.9635
	RJ	68	34471	33724	127	729	1.7186
	RW	58	34483	33715	154	768	1.5789
	FD	64	34458	33722	147	736	1.6685
	BT	59	34339	33651	218	688	1.5782
	3OT	64	33557	33189	680	368	1.9803
Average							1.748
Irreg	IPAN	76	28930	28012	434	918	3.6121
	RJ	84	28708	27831	615	877	4.4058
	RW	60	28669	27705	741	964	3.5962
	FD	82	29448	28325	121	1123	3.5860
	BT	96	29350	28377	69	973	3.5165
	3OT	79	28940	28178	268	762	2.8605
Average							3.5962
Pige	IPAN	74	41676	40655	48	1021	1.9434
	RJ	82	41540	40617	86	923	2.0327
	RW	75	41659	40685	18	974	1.8278
	FD	109	41762	40686	17	1076	2.9269
	BT	83	41311	40431	272	880	2.3491
	3OT	85	40902	40514	189	388	1.2049
Average							2.0475

Plane	IPAN	80	21383	20289	219	1094	5.1219
	RJ	93	21107	19972	536	1135	7.5776
	RW	98	21327	20288	220	1039	6.0162
	FD	114	21632	20362	146	1270	7.8712
	BT	78	21307	20062	446	1245	6.4315
	3OT	76	20495	20149	359	346	2.6126
Average							5.9385
Circles	IPAN	55	34472	33718	125	754	1.4285
	RJ	55	34683	33843	0	840	1.3651
	RW	53	34790	33843	0	947	1.4830
	FD	54	34562	33840	3	722	1.1568
	BT	57	34548	33836	7	712	1.2109
	3OT	59	33716	33484	359	232	1.0303
Average							1.2791
Stick	IPAN	44	5699	5304	40	395	3.5815
	RJ	53	5748	5312	32	436	4.6414
	RW	54	5813	5343	1	470	4.7593
	FD	34	5750	5283	61	467	3.3592
	BT	36	5697	5267	77	430	3.4154
	3OT	39	5025	4974	370	51	3.0724
Average							3.8049
Bubble	IPAN	59	43782	42808	10	974	1.3558
	RJ	53	43784	42811	7	973	1.2130
	RW	55	43802	42786	32	1016	1.3461
	FD	63	43341	42617	201	724	1.360
	BT	55	43397	42663	155	734	1.1419
	3OT	43	42489	42410	408	79	0.4890
Average							1.1510

Table 2. Comparison of the methods applied to the eight contour shapes

Method:	IPAN	RJ	RW	FD	BT	3OT
Average ε:	2.5474	3.0045	2.7001	2.9267	2.6097	1.7769
σ:	1.3933	2.3222	1.8627	2.2143	1.8210	0.9853

Table 3. The Average error rate and the standard deviation of the methods to reconstruct the contours.

Given the data distribution of each method, we also computed the standard deviation s. The standard deviation of each method is presented in Table 3. In Fig. 14, we can observe how dispersed the data are regarding their average values.

The standard deviation tells us that the RJ values are the most dispersed, whereas 3OTs are the least. We think this is due to the nature of the method to work on the variability in the contour.

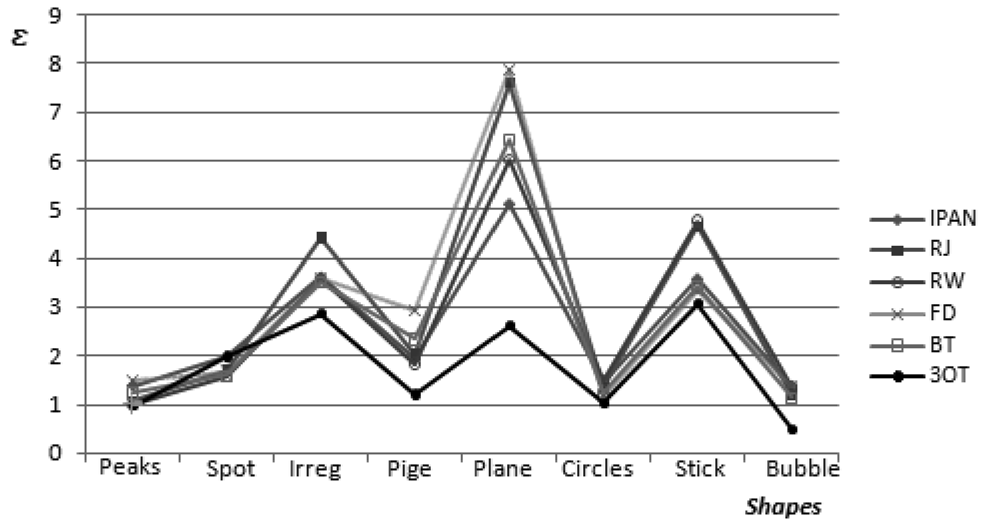


Figure 14. Error obtained by each method in polygonal approximation

From Fig. 14 we can observe an interesting situation. Most of the methods do not have such a good performance on, either, irregular shapes or shapes with nearby corner points in their neighborhood, such as the Plane, Irreg and Stick. On the contrary, they work better on smoother contours, or shapes with not so nearby corner points. For example, RJ is more efficient with smooth contours than with irregular ones, making the error smaller in the first case. On the contrary, 3OT behaves better all along the shapes, in comparison with the rest of the methods. These observations suggest that the 3OT code is more appropriate to represent shapes and to be used for looking for patterns of symbols representing shape characteristics.

We have compared different corner detectors. They are related with the chain code used to represent the contours. We have decided to use the 3OT code instead of the Freeman code because of its versatility to handle few symbols in order to find patterns; also,because of its advantage in compression efficiency. Comparisons in shape representations involving 3OT, Freeman codes and others are presented in [2] and [22].

5. Conclusions

We proposed searching pattern substrings to look for corner points. This method has three parameters: the length of elemental contour strings, the number of a symbol that can be repeated in the elemental contour strings, and the distance between pivot symbols. When decreasing one or more of these parameter values, more corner points can be found.

In order to save time and memory storage of contours, we used three symbols that represent changes of direction when covering the contours of the binary shapes. We used three classes of pattern substrings to obtain the most important corners in contour shapes, preventing to compute angles and curvatures directly.

Part of our method was to find a polygonal approximation to the original contour, an error was measured and we have found that the best approximation corresponds to apply the proposed 3OT method.

We have presented a new research strategy involving avoiding computing explicitly angles and curvatures and, also, using a recent chain code method to represent contour shapes. As future work, it would be interesting to study whether this method is invariant under scaling.

References

[1] F. Attneave. Some information aspects of visual perception. Psychol. Rev. 61 (1954) 183-193.

[2] H. Sánchez-Cruz; R. M. Rodríguez-Dagnino. Compressing bi-level images by means of a 3-bit chain code. Optical Engineering. SPIE. 44 (9) (2005) pp 1-8. 097004.

[3] H. Freeman and L. S. Davis, A Corner-Finding Algorithm for Chain-Coded Curves. IEEE Trans. Comput. 26: (1977) 297-303.

[4] C-H. Teh, and R.T. Chin, On the Detection of Dominant Points on Digital Curves. IEEE Trans of Pattern Anal and Mach Int. 11 (8) (1989) 859-872.

[5] Hong-Chih Liu; M.D. Srinath. Corner Detection From Chain-code. Pattern Recognition. 23 (1/2) (1990) 51-68.

[6] G. Medioni; Y. Yasumoto. Corner detection and curve representation using cubic B-Splines. Comput. Vision Graphics Image Process. 39: (1987) 267-278.

[7] H.L. Beus; S.S. H. Tiu. An improved corner detection algorithm based on chain-coded plane curves. Pattern Recognition. 20 (1987) 291-296.

[8] A. Rosenfeld; E. Johnston. Angle detection on digital curves. IEEE Trans Comput. 22 (1973) 875-878.

[9] A. Rosenfeld; J.S. Weszka. An improved method of angle detection on digital curves. IEEE Trans. Comput. 24: (1975) 940-941.

[10] F. Cheng; W. Hsu. Parallel algorithm for corner finding on digital curves. Pattern Recognition Lett. 8: (1988) 47-53.

[11] H. Freeman. On the Encoding of Arbitrary Geometric Configurations, IRE Trans. on Electr. Comp. 10 (2) (1961) 260-268.

[12] W. Wen-Yen. An adaptive method for detecting dominant points. Pattern Recognition. 36 (2003) 2231-2237.

[13] A. Sobania and J.P.O. Evans. Morphological corner detector using paired triangular structuring elements. Pattern Recognition. 38 (2005) 1087-1098.

[14] J. Basak and D. Mahata. Connectionist Model for Corner Detection in Binary and Gray Images. IEEE Trans. on Neural Net. 11 (5) (2000) 1124-32.

[15] J. Koplowitz, S. Plante: Corner detection for chain coded curves. Pattern Recognition 28 (6) 843-852 (1995)

[16] S. H. Subri, H. Haron, R. Sallehuddin, Neural Network Corner Detection of Vertex Chain Code. AIML Journal. 6(1) 2006 37-43.

[17] F. Arrebola, F. Sandoval. Corner detection and curve segmentation by multiresolution chain-code linking. Pattern Recognition 38 (2005) 1596 – 1614.

[18] M. Marji; Pepe Siy. Polygonal representation of digital plannar curves through dominant point detection-a nonparametric algorithm. Pattern Recognition 37 (2004) 2113-2130.

[19] E. Bribiesca. A chain code for representing 3D curves. Pattern Recognition. 33(5)(2000),755-765.

[20] H. Sánchez-Cruz. A Proposal Method for Corner Detection with an Orthogonal Three-direction Chain Code. Lecture Notes in Computer Science. Springer Berlin/Heidelberg. Volume 4179 (2006) pp 161-172.

[21] D. Chetvarikov, Z. Szabo. http://visual.ipan.sztaki.hu/corner/corner_click.html.

[22] S. Alcaraz-Corona, R. A. Neri-Calderón and R. M. Rodríguez-Dagnino Efficient bilevel image compression by grouping symbols of chain coding techniques Optical Engineering 48(3), 037001 (March 2009)

Authors Biography

Hermilo SÁNCHEZ-CRUZ

He received his PhD in sciences (computing) from the Universidad Nacional Autónoma de México (UNAM) in 2002. He received his BSc in physics from UNAM in 1995. He is a full-time professor with the Universidad Nacional Autónoma de Aguascalientes in Mexico (UAA) where he teaches graduate courses in pattern recognition and image processing. He was an assistant researcher with the Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) at the UNAM where he took part in projects about biomedical images and also recognition of Mesoamerican images. His areas of interest are pattern recognition, image compression, bidimensional and 3-D image recognition.

Ernesto BRIBIESCA

He received his BSc degree in electronics engineering from the Instituto Politécnico Nacional in 1976 and his PhD degree in mathematics from the Universidad Autónoma Metropolitana (UAM) in 1996. He was researcher at the IBM Latin American Scientific Center and at the Dirección General de Estudios del Territorio Nacional (DETENAL). He is associate editor of the Pattern Recognition journal. He has twice been chosen Honorable Mention winner of the Annual Pattern Recognition Society Award. He is currently a professor with the Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) at the Universidad Nacional Autónoma de México (UNAM) where he teaches graduate courses in pattern recognition.