Usefulness of Boundary Sequences in Computing Shape Features for Arbitrary Shaped Regions

Seongok Kim
Korea Institute of Footwear
& Leather Tech., Pusan, Korea
sokim@kiflt.re.kr

Sungyoung Kim
Dept. of Multimedia
Changwon-College, Changwon, Korea
sykim@changwon-c.ac.kr

Jongmin Kim
Division of Computer Science
Kosin Univ., Pusan, Korea
jmkim@kosin.ac.kr

Minhwan Kim
Dept. of Computer Engineering
Pusan National Univ., Pusan, Korea
mhkim@pusan.ac.kr

Abstract

A boundary sequence is a good representation of arbitrary shaped regions, but not directly used in computing shape features such as area, centroid, orientation, and so forth. In this paper, we show that the shape features can be easily computed by using cross-sections derived from a boundary sequence. The cross-sections are vertical line segments in the region and can be determined by tracing the boundary sequence once. Furthermore, a boundary sequence extraction method is also proposed, which generates a boundary sequence for each region in a binary image by scanning the image only once. The proposed method works well even if a region has holes.

1. Introduction

Shape representation is an important problem in image processing and pattern recognition. A good shape representation makes it easier for a shape to be stored, transmitted, compared against, and recognized. Several techniques for shape representation have been developed [1], which could be classified into two categories, area-based type (e.g., quad-tree, run-length encoding, skeleton, shape decomposition methods) and contour-based one (e.g., chain code, Ψ-s plot, slope density function, approximation methods).

There are several criteria for a good shape representation [1]:

- Efficiency: simplicity and compactness
- Accuracy: accurate and complete reconstruction
- Effectiveness: suitability for shape analysis and shape recognition

In [2], additional criteria are also discussed, e.g., uniqueness, stability under noise, allowance of abstraction, invariance under geometric transformation, fast computability, and well defined mathematical characterization. However, it is difficult to find the good representation that satisfies all the above criteria.

Recently, morphology-based methods [2, 3] for shape representation have been extensively studied. They satisfy the efficiency and the accuracy criteria well, but they suffer from computing shape features that are useful for shape analysis and recognition. They also suffer from a lot of computation. The run-length encoding can be another good representation, because it is a compact, accurate representation and provides efficient computation of shape features [1, 4]. However, it is not suitable for shape analysis and recognition, because it does not provide intrinsic characteristics of a given shape well. On the one hand, the contour-based representations generally suffer from representing of contours accurately [1] and computing shape features.

In this paper, we try to rediscover usefulness of a boundary sequence as a contour-based shape representation. The boundary sequence is defined in this paper as an ordered list of boundary pixel locations in clockwise order. The boundary sequence can be a simple, compact representation for an arbitrary shape and can describe contour of the shape accurately and completely. However, the boundary sequence is not suitable for computing shape features. Therefore, a cross-section generation algorithm is proposed in section 2, which determines all the cross-sections in a given shape by tracing boundary sequence of the shape once. The cross-

1051-4651/02 $17.00 (c) 2002 IEEE

sections are similar to run-lengths in the shape, so shape features can be easily computed (Section 3) as in the run-length encoding representation. The boundary sequence is also useful in computing minimum bounding rectangle(MBR), perimeter, Euler number of the shape that cannot be computed easily in the run-length encoding representation. A boundary sequence extraction method is also proposed in section 4, which generates a boundary sequence for each shape in a binary image by scanning the image only once. The cross-section generation algorithm and the boundary sequence extraction method work well even if a shape has holes.

2. Cross-section generation algorithm

A shape is represented as a region R in a digital binary image and a clockwise boundary sequence of the region is defined as follows:

$$R = \{(x, y) | (x, y) \in \text{ shape } \}$$
$$BS(R) = \{(x_u, y_u), u = 0, 1, \ldots, n-1\}$$

where $|x_k - x_{k-1}| \leq 1$ and $|y_k - y_{k-1}| \leq 1$ for $k \geq 1$. A vertical cross-section, $C(x_i, y_l : y_u)$ can be defined as

$$C(x, y_l : y_u) = \left\{ (x, y) \;\middle|\; \begin{array}{c} y_l \leq y \leq y_u, (x, y) \in R, \\ (x, y_l) \text{ and } (x, y_u) \in BS(R) \end{array} \right\},$$

where (x, y_l) and (x, y_u) represent the lower and the upper end-points of the vertical cross-section, respectively.

When a boundary sequence is traced sequentially, there are three types of x-value difference (Δx) between a current point and its previous one. If $\Delta x > 0$, the trace from the current point to the previous point is called a *right-trace*, or we say that the trace is a trace *with positive* Δx. If $\Delta x < 0$, the trace is called a *left-trace*. If $\Delta x = 0$, the trace with positive (resp. negative) Δy is called a *down-trace* (resp. an *up-trace*). If consecutive traces consist of right- or left-traces, there are only six cases as shown in Fig. 1.

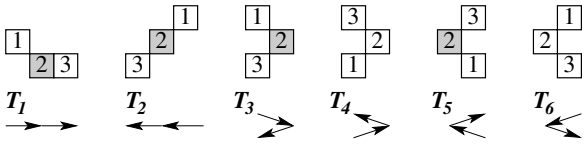

Figure 1. Six cases of consecutive traces.

The T_1 and T_2 in Fig. 1 represent cases of not changing the direction of traces. Then the middle point p_2 in T_1 (resp. T_2) is registered as the upper (resp. lower) end-point of a vertical cross-section as shown in Fig. 3.

The T_3, T_4, T_5, and T_6 represent typical cases of changing the direction of traces. In case of changing in a clockwise fashion, e.g., T_3 and T_5, the rightmost point (p_2 in T_3) or the leftmost point (p_2 in T_5) is a vertical cross-section in itself. However, the extreme points in T_4 and T_6 are determined as the middle points in a vertical cross-section, because the trace direction changes in a counter-clockwise fashion. Whether the trace direction changes in a clockwise or a counter-clockwise fashion can be easily determined by testing difference of y-values between previous and next point of the extreme point. For example, the difference in T_3 is greater than 0, while that in T_4 is negative.

When there are down-traces or up-traces between consecutive right- or left-traces, there is a vertical run (a, b in Fig. 2). If the previous trace is a right-trace, top point of the vertical run is registered as the upper end-point (a in Fig. 2). If it is a left-trace, bottom point is registered as the lower end-point (b in Fig. 2). When there is a vertical run in case of T_3 or T_5, its top and bottom points are registered as the upper and the lower end-points, respectively.

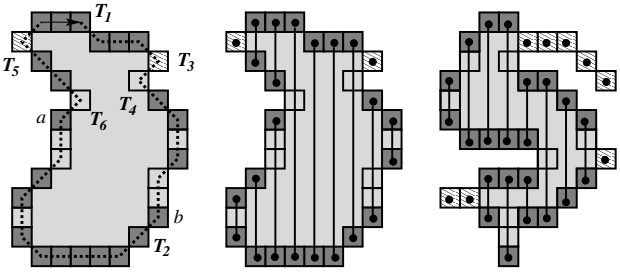

Figure 2. Examples of cross-sections.

The vertical cross-section generation algorithm is summarized as follows:

// input : a boundary sequence for a region R,
$BS(R) = \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$

1. Trace $BS(R)$ until a point (x_i, y_i) whose x-difference Δx_i $(= x_i - x_{i-1}) \neq 0$ is found. If there is not such a point, Insert-CS-list($x_0, \min\{y_i | (x_i, y_i) \in BS(R)\}$) and Insert-$CS$-list($x_0, \max\{y_i | (x_i, y_i) \in BS(R)\}$). Then, exit.
2. Set the direction flag $F_d = \Delta x_i$ and the starting point $s = (x_i, y_i)$. Set $Y_{pre} = y_i$ and $Y_{current} = y_i$.
3. Set $j = (i+1) \bmod n$ and move to next point (x_j, y_j). Compute $\Delta x_j (= x_j - x_i)$.
   - 3.1 If $\Delta x_j = F_d$, Insert-CS-list($x_i, Y_{current}$). Then set $Y_{pre} = Y_{current}$ and $Y_{current} = y_j$.
   - 3.2 If $\Delta x_j = -F_d$, then
     - 3.2.1 If $F_d \cdot (Y_{pre} - y_j) \geq 0$, Insert-$CS$-list($x_i, Y_{current}$) and Insert-$CS$-list($x_i, y_i$). Then, set $Y_{pre} = y_i$, $Y_{current} = y_j$, and $F_d = \Delta x_j$.
     - 3.2.2 Else, set $Y_{current} = y_j$ and $F_d = \Delta x_j$.
   - 3.3 If $\Delta x_j = 0$ and $F_d = \Delta y_j$, then $Y_{current} = y_j$.
4. Repeat step 3 until $(x_j, y_j) = s$.

Step 1 in the algorithm is to find a starting point whose previous pixel is to its left or right. For a vertical shape with one-pixel width, the algorithm outputs only one vertical cross-section.

The CS-list consists of $(x_{max} - x_{min} + 1)$ buckets. A bucket[x] keeps y-values of end-points of vertical cross-sections $C(x, y_l : y_u)$'s being sorted in ascending order. The procedure Insert-CS-list(x, y) inserts y into appropriate location among y-values in bucket[x] as shown in Fig. 3. Eventually, we can determine vertical cross-sections by pairing the y-values of each bucket[x]. In Fig. 3, there are three vertical cross-sections, $C(a, 3 : 7)$, $C(a, 9 : 11)$, and $C(a, 13 : 16)$.

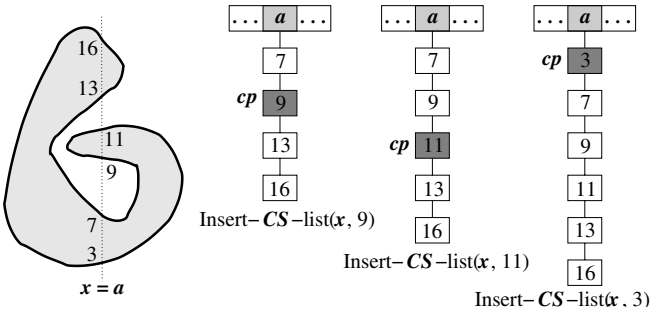

Figure 3. Examples of insert-CS-list(x, y).

If a hole H is in a region R, we can determine vertical cross-sections by merge-sorting y-values of bucket[x] in CS-list for R and those in CS-list for H. However, in CS-list for H, y_u should be increased by 1 and y_l be decreased by 1, because $BS(H)$ is the boundary sequence that is obtained by following the boundary of the hole.

3. Shape features from cross-sections

Let the set of vertical cross-sections for a region R be Ω. Then, the region can be represented as the union of the vertical cross-sections in Ω.

$$R = \bigcup_{C(x,y_l:y_u)\in\Omega} C(x, y_l : y_u).$$

Let define the length of a vertical cross-section $C(x, y_l : y_u)$ as follows.

$$l(x, y_l : y_u) = y_l - y_u + 1.$$

We can easily compute perimeter, minimum bounding rectangle, aspect ratio of a region R from its boundary sequence. The Euler number is defined as the number of components minus the number of holes. So it can be also determined easily from the number of the holes, $N(H)$ in R.

$$E = 1 - N(H)$$

Area of the region can be computed by summing lengths of vertical cross-sections as in the run-length encoding representation [1, 4]. Compactness [1] or thinness [5] is also determined easily, which is defined as (perimeter)^2/(area). However, centroid of the region can be determined directly as follows, while the vertical projection should be computed previously in run-length encoding representation.

$$\bar{x} = \frac{1}{A}\sum_{C(x,y_l:y_u)\in\Omega}(x \cdot l(x, y_l : y_u))$$
$$\bar{y} = \frac{1}{2A}\sum_{C(x,y_l:y_u)\in\Omega}((y_l + y_u) \cdot l(x, y_l : y_u))$$

The vertical projection of the region can be computed as follows by summing vertical cross-sections in each bucket.

$$V[i] = \sum_{C(x,y_l:y_u)\in\Omega \& x=i} l(x, y_l : y_u)$$

Horizontal and diagonal projection can be also computed by using similar technique described in [4]. Another method is to use horizontal cross-sections or diagonal cross-sections that are determined in similar ways as in section 2.

Orientation of an elongated shape is defined as orientation of the axis of least inertia in [1, 5] and it can be computed from three parameters, a, b, and c as follows:

$$a = \sum_{(x,y)\in R}(x - \bar{x})^2 = \sum_{C(x,y_l:y_u)\in\Omega}(x - \bar{x})^2 \cdot l(x, y_l : y_u),$$

$$b = \sum_{(x,y)\in R}(x - \bar{x})(y - \bar{y}) = \sum_{C(x,y_l:y_u)\in\Omega}(x - \bar{x}) \cdot f(n_u, n_l), \text{ and}$$

$$c = \sum_{(x,y)\in R}(y - \bar{y})^2 = \frac{1}{6}\sum_{C(x,y_l:y_u)\in\Omega}(x - \bar{x}) \cdot g(n_u, n_l),$$

where,

$$f(n_u, n_l) = |n_u|(|n_u| + 1) - |n_l|(|n_l| + 1),$$
$$g(n_u, n_l) = |n_u|(|n_u| + 1)(2|n_u| + 1) - |n_l|(|n_l| + 1)(2|n_l| + 1),$$
$$n_u = y_u - \bar{y}, \quad \text{and} \quad n_l = y_l - \bar{y}.$$

That is,

$$\tan(2\theta) = b/(a - c).$$

4. Boundary sequence extraction method

From above discussions, we know that a region can be described effectively by its boundary sequence. If a region R includes holes, $H_1, \ldots, H_m$, it should be represented as a boundary descriptor, $BD(R)$, defined as follows.

$$BD(R) : BS(R) \rightarrow BS(H_1) \rightarrow \ldots \rightarrow BS(H_m)$$

The boundary sequence extraction method scans the image in the raster-scan manner as in TV. There are three different states of current scan- pixel, p in a scan line as follows and the state transition diagram can be represented as in Fig. 4.

S_0 : $p \in$ background region
S_1 : $p \in$ foreground region
S_2 : $p \in$ hole region

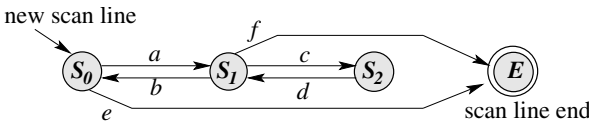

Figure 4. State transition diagram.

The state transition diagram is initialized when a new scan line starts and finishes at location of end-pixel in the scan-line (state E). When a new object region or a new hole region is being extracted, pixels of extracted boundary sequence are marked with appropriate label that is assigned differently to each object region and each hole region. The boundary sequence can be extracted by the boundary following operation [1]. A nested-hole counter is used to determine which state is the next one when a background pixel is found in S_1. The transition conditions are summarized as follows.

a, d : When a foreground or a labeled pixel is met.
b : When a background pixel is met and the nested-hole count is zero.
c : When a background pixel is met and the nested-hole count is not zero.
e : When there remain only the background pixels in the current scan line.
f : When no background pixel is found.

In each state, following operations are performed.

S_0 : When a foreground pixel is found, a new boundary descriptor for the new object region is generated through boundary following.
S_1 : When a background pixel is found, a new hole boundary sequence for the new hole region is generated through boundary following. The hole boundary sequence is attached to appropriate boundary descriptor.
S_2 : When a foreground pixel is found, a new boundary descriptor for the new object region in a hole is generated.

Fig. 5 shows examples of extracting boundary sequences in an image. When all pixels in a scan-line are background pixels, the transition condition e occurs. If a foreground pixel is found in S_0, then a new boundary descriptor is generated through following the boundary of corresponding object region (R_1), as shown in Fig. 5(a). The transition condition b occurs in Fig. 5(b) when a background pixel is met in S_1 and the nested-hole counter is zero, while the transition condition c occurs in Fig. 5(d) because the nested-hole counter is not zero. The hole (H_1) is detected in Fig. 5(c), because a background pixel is found in S_1. A new boundary descriptor for object region (R_3) is also extracted, because a foreground is found in S2. Three boundary descriptors extracted in Fig. 5 are as follows.

$BD(R_1) : BS(R_1)$
$BD(R_2) : BS(R_2) \rightarrow BS(H_1)$
$BD(R_3) : BS(R_3)$

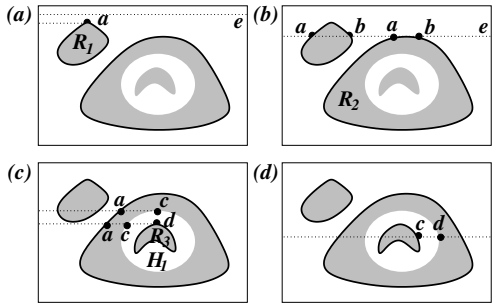

Figure 5. Examples of boundary sequences.

5. Conclusions

We showed that shape features could be efficiently computed by using the cross-sections derived from a boundary sequence. Therefore, the boundary sequence can be a good shape representation. A one-pass method is also proposed to generate a boundary sequence for each region in a binary image, which can be considered as a one-pass connected component labeling method.

References

[1] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, McGraw-Hill, Singapore, 1995.

[2] I. Pitas and A. N. Venetsanopoulos, "Morphological shape decomposition," *IEEE Trans. on PAMI* 12(1):38–45, 1990.

[3] J. M. Reinhardt and W. E. Higgins, "Comparison between the morphological skeleton and morphological shape decomposition," *IEEE Trans. on PAMI* 18(9):951–957, 1996.

[4] B. K. P. Horn, *Robot Vision*, MIT Press, London, 1986.

[5] S. E. Umbaugh, *Computer Vision and Image Processing*, Prentice-Hall, London, 1998.