Abstract

We present a novel approach to the coding and representation of shapes using intersections. Our algorithm combines contour and region based approaches. After exploring hitherto unvisited regions of the shape, we find salient features on the boundary in that region. Our method has a number of attractive features. First, it is efficient to compute, since only small yet salient regions are retrieved and analysed. Second, our algorithm generates a contour and region analysis simultaneously. Third, the algorithm is fully adaptive, so we do not need to set parameters, such as the number of features to be found, in advance. Finally, the algorithm can be implemented using familiar data structures including the max heap, circular linked lists and trees. We demonstrate the stability of the algorithm with a few practical image sets and developed an extended version which deals with enclosed shapes.

1. Introduction

As the number of digital images increases ever more rapidly, so does the requirement to search huge databases become more pressing. Given a search item, it is particularly important to be able to explore growing databases for similar images or objects. The key to searching such large databases is an appropriate representation (hence coding) of shape. However, there are two major issues in developing a suitable shape representation. One is the difficulty of extracting relevant and precise information about the shape, since images are generally distorted or projected to a lower dimension (typically 2D or 3D), so the shape often conveys lossy information. The other is more practical and includes consideration of whether and how to maintain the image information with a small amount of memory and how to structure it for efficient and effective search.

A large number of shape coding and representation techniques have been developed. Though a brief survey is inevitably crude, it is reasonable to assert that there are two major approaches to shape representation and description: contour-based and region-based. In some cases, such as the medial axis transform, and developments of it, there are duality relationships, in the sense that one can compute the contour representation from the region representation, and vice versa. In each of these two classes, methods can be further sub-divided into structural and global methods. This amounts to deciding whether the shape is represented as a whole or represented by partial segments. The whole hierarchy of such representation is well described in [25].

The global contour-based approach to shape calculates and uses a multi-dimensional feature vector derived from the entire boundary in order to describe the shape. A typical such approach uses simple global contour shape descriptors [24, 17], such as: area, circularity (perimeter²/area), eccentricity (length of major axis or length of minor axis), major axis orientation, bending energy, convexity, ratio of principle axis, circular variance and elliptic variance. Another global contour-based approach is based on correspondence matchings which measures the similarity between shapes using point-point matching to calculate, for example, the Hausdorff distance between them [18, 2]. Shape signature is also a global method of contour based shape representation [8]. Yet other global contour-based approaches use: boundary moments [20]; Elastic matching [4]; scale space analysis [1]; and spectral transforms [23].

Structural contour-based approaches divide shapes into boundary segments, which we may call “primitives”. Such methods include: chain codes; polygon decompositions; smooth curve decompositions; scale space methods; and a variety of shape invariants. Chain codes represent a shape as a sequence of unit vectors to describe the shape [9]. Shape boundaries can also be decomposed using a polygonal approximation (with an associated tree structure) [12] or using B-splines [15]. Zero-crossings of curvature, computed at a range of scales as defined by a Gaussian smoothed boundary, have been advanced as a set of primitives and used for model matching [3]. Shape invariants, which include geometric invariants such as cross-ratio, length ratio, distance ratio [20], algebraic invariants such as determinant, eigenvalues and differential invariants such as curvature, and in-
tegral invariants. These methods have been shown to be powerful for partial matching of shapes. Nevertheless, they have a number of limitations. First, in some cases it is difficult to generate primitives if their number is not known in advance. Second, such methods are typically sensitive to noise. Third, it can be computationally costly to search for the sub-shape matching. Finally, one can fail to find global features by only focusing on partial structures.

Region-based shape representations can also be subdivided into global and structural approaches. Global region-based shape approaches include: geometric moment invariants [13]; algebraic moment invariants [21]; orthogonal moments based on Legendre and Zernike polynomials [22]; grid-based methods [14]; and methods based on partial differential equation analyses of shapes, most recently based on the Poisson Equation [10]; and the shape matrix [11]. Since global region-based methods measure the pixel distribution of the shape region, they are less affected by noise and minor variations. Finally, structural region-based approaches include: the convex hull and Medial axis transform[20, 19].

In this paper, we present a shape representation that combines the best features of both contour-based and region-based approaches. The paper is organized as follows. In section 2, we describe the framework for our proposed shape representation algorithm. Then, we demonstrate the stability of the representation and clustered features in section 3.

2. Proposed algorithms

Our algorithm is iterative: as hitherto unvisited regions of the shape are explored, salient features are found on the contour. As we will see, the encoding of our algorithm is an efficient adaptive and iterative process.

2.1. Algorithms

Let \( I \) denote an image whose size is \( T_1 \times T_2 \). Suppose that we have a graph \( G = (\{E, H\}, \{C, N\}) \) where \( \{E, H\} \) and \( \{C, N\} \) are the sets of edges and nodes. The set of edges \( \{E, H\} \) consists of two different kinds of edges: internal edges \( H \) and leaf edge \( E \). The set of vertices \( \{C, N\} \) also comprises two different types: \( c_i \) and \( n_j \). As we explain below, \( c_i \) is a node for the \( i \)th centre which is used for the skeleton of the shape; whereas \( n_j \) is the \( j \)th leaf node on the boundary of the shape. Because the internal edges \( H \) is expressed implicitly by the structure of central nodes \( C \), we do not have to build an explicit structure for \( H \). That is, it is sufficient to use only three sets to define the graph: \( E \), the leaf edges; \( N \), the boundary nodes; and \( C \), the central nodes as shown in Fig. 1.

![Figure 1. Graph: \( N = \{n_1, n_2, \ldots, n_5\} \) is the set of leaf nodes and \( E = \{e_1, e_2, \ldots, e_5\} \) is the set of leaf edges which connect leaf nodes and central nodes. \( C = \{c_1, c_2, c_3\} \) is the set of central nodes. \( H = \{h_1, h_2\} \) is a set of internal edges.](image)

2.1.1 Main algorithms

Initially, an arbitrary point is chosen inside the object. The choice may either be made manually or it can be computed automatically, based, for example, on the Jordan curve theorem [16]. The insensitivity of our method to this initial choice will be discussed later. This is a central point, \( c_1 \). The firing of intersections from \( c_1 \) generates \( k \) vertices, on the contour (boundary), which are connected to the central nodes where \( k \geq 3 \). The Bresenham algorithm for line drawing is employed to obtain intersections [5]. The boundary points are then found from the intersections detected by using first or second derivatives. The \( k \) points are denoted \( n_1, n_2, \ldots, n_k \). In this paper, for simplicity, we set \( k \) be 3 as depicted in Fig. 2.

![Figure 2. Three nodes after the first step of the algorithm](image)

The remainder of the algorithm proceeds recursively. The longest edge \( \ell_{ij} \) is selected. We then define a new center \( c_2 \) at the midpoint of \( c_1 \) and \( n_j \). Here, \( n_j \) is used as a control node to calculate two angles for intersections. We obtain two neighbours of \( n_j \) from \( N \): a left node \( n_l^j \) and a right node \( n_r^j \). We then calculate two angles \( \angle n_l^j c_2 n_j \) and \( \angle n_r^j c_2 n_j \). Fig. 3 shows the second processing step for the example shape and \( \angle n_l^j c_2 n_j \) and \( \angle n_r^j c_2 n_j \) are replaced by \( \angle n_1 c_2 n_3 \) and \( \angle n_3 c_2 n_1 \) respectively.

Two intersections \( \ell_{j1} \) and \( \ell_{j2} \) are drawn from \( c_2 \) with the two angles and a control node. The Bresenham line
As a result of several such iterations, we can monitor the changes on the graph from the 1st to the 9th step as in Fig. 5. Following the 9th processing step, we can graphically decompose the shape into a reconstructed boundary and a backbone (skeleton) with leaf edges as shown in Fig. 6. As the graphs shown in Fig. 6 are obtained, several questions arise. The first asks how many recursive iterations are required to get a "sufficient" shape representation (where, of course, "sufficient" is only meaningful relative to a particular application). The second asks whether or not the approach is stable, in the sense that a minor change in the shape or the initial seed point’s position, materially affect the final shape representation. Third, we need to consider the scope of the representation: what shapes can be represented effectively by our algorithm. As we will see, a particular instance of this latter question is the ability to represent enclosed shapes.

First, the algorithm can be terminated in various ways. It may, for example, halt the iterative refinement of the representation by restricting the length of the edge. More precisely, if the length of the longest edges of \( E \) is smaller than some pre-specified length \( D_{e,n}^{\min} \), the algorithm terminates. However, in practice, it may not be necessary to iterate the algorithm until the length of the longest edge is smaller than \( D_{e,n}^{\min} \). For example, if we capture the important features properly, a small number of points can suffice to express the shape. This can, for example, be quantified by the similarity between the original shape and the shape that is reconstructed from the representation. Let \( I \) and \( I_r \) be the original image and the reconstructed polygon with small number of features. The similarity for a silhouette is calculated by

\[
S(I, I_r) = \frac{I_{in} \cap I_{r, in}}{I_{in} \cup I_{r, in}^r}
\]

where \( I_{in} \) and \( I_{r, in} \) stand for the ‘inside’ and the ‘outside’ of the shape, regarded as a silhouette. \( S(I, I_r) \) has a value between 0 and 1. As the similarity becomes close to 1, the reconstructed polygon becomes increasingly close to the original image. Such a similarity measure can be calculated at each iteration, and if it is larger than a given threshold, such as 0.95, the algorithm is terminated. Importantly, we do not need prior knowledge about, for example, the number of features, rather we can obtain such information adaptively.

It is straightforward to address the second question of stability using the similarity measure given in Eq. (1). The reconstructed polygons from several runs are compared to the original image. The results of the proposed algorithm are stable, both in volume reconstructed with features, and in the number of features if the similarities of Eq. (1) are always satisfied.

The third question concerns whether all kinds of shapes can be represented adequately using our algorithm. Fig. 7, which is another view of Fig. 6, describes the fact that there are two types of mis-matched regions: approachable (convex: blue solid line) and unapproachable regions (concave:...
red solid line). If the proposed algorithm can detect easily the pertinent features in an interesting region, we call that region "approachable". Otherwise, we call it "unapproachable". While approachable regions can be merged to the reconstructed polygon, perhaps with more iterations, it is difficult to find the nodes in the unapproachable regions with the proposed algorithm since the angle for possible new nodes is close to zero. We therefore need extra processing to address this issue.

![Figure 7. Two types of mis-matched regions: approachable region (blue solid lines) and unapproachable region (red solid lines) where red dotted line denotes the reconstructed polygon.](image)

To find features in unapproachable regions, we first find two nodes with a long path between them. For example, the nodes $n_4$ and $n_6$ have a path whose length is 5 which is longer than other nodes. We make the path shorter by firing one more intersection between the two nodes from $c_3$. Another intersection for $n_5$ and $n_{18}$ is also drawn in a same way as in Fig. 8. Then, the algorithm is run, recursively, again.

![Figure 8. Finding nodes of unapproachable regions.](image)

In this paper, we address the accessibility using shapes that are formed by embedding one inside another. Such embedded shapes cause difficulties for a number of shape representation schemes; but a fuller discussion of this point is beyond the scope of this article. In shape representation, one of the interesting issues is to code enclosed shapes as shown in Fig. 9-(a) and -(d). Currently, our algorithm may have some difficulty addressing this issue directly. In order to reduce this shortcoming, we need a pre-processing algorithm to check whether there are enclosed shapes before firing intersections. To date, we have used the Jordan Curve Theorem to classify the interior shapes from outer shapes. First, we find intersections at all pixel points with four directions, $0$, $\pi/2$, $\pi$, and $3\pi/2$. In each direction, we calculate the number of change points using first and second derivatives and we store the minimum number of change points in a map. Fig. 9-(b) and -(e) are the constructed maps. However, there are still some areas of the map which are not properly estimated. We re-estimate by merging the regions which have the same values in a modular operation by 2. Finally, we obtain the modified map as shown in Fig. 9-(c) and -(f). After finding this map, we apply our algorithm from the higher value areas to the lower value areas sequentially. For example, the inner area (map value=3) of the eye is simulated in the bird image. Afterwards, outer (map value=2) of the eye and the leg section are retrieved. Then, the whole body (map value=1) is explored by our algorithm to code the shape. Note that when there are enclosed shapes, it cannot be analysed or represented using a single data structure. In other words, if an image is decomposed into $k$ enclosed images ($k=3$ for the bird image and $k=1$ for the helicopter image), we need $k+1$ double circular linked lists, $k+1$ max-heaps, and $k+1$ trees which we will explain the details.

2.2. Data Structures for implementation

The data structures used to implement $E$, $N$ and $C$ are respectively a max-heap, a circular double linked list, and a tree [7]. More precisely, we have the following conditions,

1. $E = \{e_1, e_2, e_3, \cdots, e_N\}$. Let $|e_i|$ denote the length of an edge, $e_i$, and $N_e$ the cardinality of elements of $E$. The data structure of this dataset is a max-Heap as shown in Fig. 10. In this figure, $|e_i| \geq |e_j|$ and $|e_i| \geq |e_k|$ and the relation is subsequently maintained in $T_j$ and $T_k$. Each edge $e \in E$ can be replaced by $(c, n)$ or $(n, c)$, where $c \in C$ and $n \in N$.

2. $N = \{n_{b_1}, n_{b_2}, n_{b_3}, \cdots, n_{b_i}, n_{b_{i+1}}, \cdots, n_{b_{N_b}}\}$ is the

![Figure 9. Enclosed shapes (a bird and a helicopter).](image)
set of boundary nodes represented by an ordered set:
\[
\begin{align*}
  n_{b_1} & \leftrightarrow n_{b_2} \leftrightarrow n_{b_3} \leftrightarrow \cdots \\
  & \uparrow \\
  n_{b_{N_b}} & \leftrightarrow \cdots \leftrightarrow n_{b_{i+1}} \leftrightarrow n_{b_i}
\end{align*}
\]
where the cardinality is \( N_b \) and \( \leftrightarrow \) denotes a double linked list.

3. \( C = \{c_{d_1}, c_{d_2}, \ldots, c_{d_{N_c}}\} \) is a set of central nodes represented by a directed tree structure. This is acyclic but we give each child centre the address of its parent so we can find all ancestors if necessary. This tree structure makes our graph work well without \( H \) and the schematic data structure of \( C \) is given by

\[
\begin{align*}
  c_{d_1} & \\
  & \downarrow \\
  c_{d_2} & \\
  & \downarrow \\
  c_{d_3} & \\
  & \downarrow \\
  c_{d_4} &
\end{align*}
\]
where the cardinality is \( N_c \). (\( H \) is the relation between the centres of the tree.)

For simplicity we will use the following terms in this paper: \textit{nodes} for \( N \), \textit{centres} for \( C \), and \textit{edges} for \( E \).

Let us return to the algorithm. After finding three nodes after the initial steps as in Fig. 2, each one is inserted sequentially into \( N \) to satisfy a circular linked list. All three edges are also inserted to \( E \). We then build a tree structure, \( C \) with \( c_1 \) and we have the data structures as in Fig. 11.

![Figure 11. Data Structures of three nodes in the first step](image)

Subsequently, we update the data structures of \( E \), \( N \), and \( C \). The data structure of \( E \) for the representation in Fig. 3 is obtained by the steps shown in Fig. 12.

Fig. 13 represents the changes to the data structures during the three 2nd, 3rd and 4th steps. The sub-figures (a), (c) and (e) of figure show the changes to the graph superimposed on the shape. The blue circles correspond to the new nodes. Cross circles are the centres and black lines are edges. The red line represents the implicit structure of the centres. The other sub-figures (b), (d), and (f) of this figure plot the changes of the data structures of \( C \), \( N \) and \( E \). The blue text in the figures show update information.

![Figure 12. The change of a max-heap structure of \( E \) during insertion of three nodes: red notations represent the new nodes which are inserted.](image)

(a) The second step (b) Data structure of the second step (c) The third step (d) Data structure of the third step (e) The fourth step (f) Data structure of the fourth step

Figure 13. The change in data structures over 3 successive steps

2.3. Algorithmic framework

The framework of the method is described in Algorithm 1. In this, there are a number of subroutines. \texttt{RetrieveHeap}(\( E \)) is a function to view the top node of the max-heap structure of \( E \). The other function \texttt{getNode}(\( c_{firing}, c_{control}, \theta \)) has inputs with a firing centre \( c_{firing} \), a control centre \( c_{control} \), and an angle \( \theta \). This func-
Algorithm 1 Intersection based Shape representation

1: $D_{\text{c}, n}^{\text{min}} = 5, s_h = 0.95, t = 1$.
2: Define a dummy control centre $c_0$ at $(0,0)$.
3: Choose manually or automatically a centre, $c_1$ in the inside of the shape.
4: Find $n_i$ by intersection algorithms with three different angles: $n_i = \text{getNode}(c_1, c_0, 2\pi \times i/3)$ for $i \in \{1, 2, 3\}$.
5: Build $E$, $N$ and $C$.
6: Calculate the similarity, $s$.
7: while $s < s_h$ do
8:   if $|\text{RetrieveHeap}(E)| < D_{\text{c}, n}^{\text{min}}$ then
9:     Do extra processing to remove unapproachable regions.
10:    If the longest edge is still shorter than $D_{\text{c}, n}^{\text{min}}$, the program terminates.
11:  end if
12:  $t = t + 1$.
13:  Pop a longest edge, $e_i$, from the max heap.
14:  Generate a centre $c_i$ on the edge.
15:  Find the neighbours of $n_i$ among $N$.
16:  Calculate angles, $\theta_0 = \angle n_i^t c_i n_i$ and $\theta_1 = \angle n_i c_i n_i^t$.
17:  Find $n_i$ by intersection algorithms with three different angles: $n_i = \text{getNode}(c_i, c_1, \theta_i-2t)$ for $i \in \{2t, 2t+1\}$.
18:  Build $E$, $N$ and $C$.
19:  Calculate the similarity, $s$.
20: end while

3. Results

We illustrate the performance of the algorithm on two types of silhouettes.

3.1. Features obtained

The first shape we consider is a starfish which has five different directional arms. Fig. 14-(a) shows three graphs: a set of nodes (left), its overlapped image (middle), and the centres for skeleton (right). As we can see in the skeleton, the centres are concentrated in the leaves and in certain specific areas. In this way, we can decompose the primitives of the shape using skeletons. The second example is a cat shape with size $380 \times 400$ of Fig. 14-(b). This figure also has three graphs: a set of nodes (left), its overlapped image (middle), and the centres for the skeleton (right).

We also tested our methods to image sets which have en-

3.2. Finding curvature

After obtaining contour features and skeletons, we clustered both centres $C$ and nodes $N$ using the mean shift al-

Figure 14. (left) Nodes, (middle) Comparison between the reconstructed image and original image, and (right) backbone closed shapes. Fig. 15 shows contour features which are detected in two different images. For the bird image, there

Figure 15. (left) Nodes, (middle) Comparison between reconstructed image and original image, and (right) backbone

Figure 16. (left) Nodes, (middle) Comparison between reconstructed image and original image, and (right) backbone
algorithm [6]. The clustering results for the four different images are shown in Fig. 17. The sub-figure (b) is for the clustering of boundary nodes \( \mathbf{N} \) and another sub-figure (c) is for the clustering of centres \( \mathbf{C} \). Sub-figure (d) shows the clustering of the union of centres and nodes. As we can see in Fig. 17 the clustering of the \( \mathbf{N} \) is prone to failure because there are too few elements to cluster and they are not well distributed for the mean shift algorithm. Also, clustering \( \mathbf{C} \) may stick to local optima with unwanted clustering. Therefore, we combined both nodes and centres and clustered them and we obtained more proper clustering modes as in Fig. 17-(d).

3.3. Stability

We simulated 100 different runs with different initial \( c_1 \)'s for images in an adaptive framework. Fig. 18 shows the changes of similarities as the iteration increases. Each run finishes at the same similarity threshold, 0.95. As we can see in this figure, the 100 runs result in similar final reconstructed polygons. This suggests that the essential number of nodes (features) are captured and the reconstructions by polygons are also stable.

4. Further work

We have shown that the proposed algorithm generates compact features which represent the shapes well. However, we have not yet applied the method for image data mining in a large database; this is our current work. In addition, we are considering applying this algorithm for a film, a set of sequential images. Last, we will extend this algorithm to coloured images.

5. Conclusions

The proposed algorithm is based on both contour-based and region-based approaches. It finds unvisited regions first by a region-based approach, then searches for the features on the boundary of the selected regions. This algorithm generates three types of well-defined and useful data structures for boundary nodes, internal nodes (centres) and leaf edges. Boundary nodes are stored in a circular double linked list and centres are maintained by tree structures. The final data structure is a max-heap structure which encodes the leaf edges. We have seen that the algorithm also provides important features from the boundary nodes and centres using clustering algorithms such as mean shift. In addition, we can obtain a skeleton of the shape from the centres. Last, this algorithm adaptively estimates the number of features.
in terms of a similarity measure.

References


