ELSEVIER

Pattern Recognition Letters 23 (2002) 731–741

Pattern Recognition Letters

www.elsevier.com/locate/patrec

An efficient shape-based approach to image retrieval

Ioannis Fudos [*], Leonidas Palios

Department of Computer Science, University of Ioannina, P.O. Box 1186, GR45110 Ioannina, Greece

Abstract

We consider the problem of finding the best match for a given query shape among candidate shapes stored in a shape base. This is central to a wide range of applications, such as, digital libraries, digital film databases, environmental sciences, and satellite image repositories. We present an efficient matching algorithm built around a novel similarity criterion and based on shape normalization about the shape's diameter, which reduces the effects of noise or limited accuracy during the shape extraction procedure. Our matching algorithm works by gradually "fattening" the query shape until the best match is discovered. The algorithm exhibits poly-logarithmic time behavior assuming uniform distribution of the shape vertices in the locus of their normalized positions. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Image retrieval; Shape-based matching; Average Hausdorff; Polylog algorithm; Geometric hashing

1. Introduction

The last few years, there is an emerging need to organize and efficiently use large pools of images that have been collected over the last decades and contain information potentially useful to areas such as medicine, journalism, weather prediction, environmental sciences, art, fashion and industry. It is estimated that there are more than 20 million pages containing hundreds of millions of images on world wide web pages alone (Carson and Ogle, 1996). Traditionally, images were retrieved by their filename, other technical characteristics such as date and size or through text keywords, in the case of manually annotated images. Manual annotation, except for being a time consuming and not real-time process, can describe only a very small percentage of the information that an image contains.

Recently, there is an increasing effort to organize and retrieve images by content based on characteristics such as color, texture, and shape. A number of methods in the literature perform indexing and retrieval based on global image characteristics such as color, texture, layout, or their combinations. QBIC (Flickner et al., 1995; IBM), a system developed at IBM Almaden supports retrieval by color histograms, texture samples (based on coarseness, contrast and directionality), and shape. QBIC uses R^* trees to process queries based on low-dimensionality features, such as, average color and texture. Shape matching is supported using either dimensionality reduction, which is sensitive to rotation, translation and scaling (Niblack et al., 1993), or by clustering using nonlinear elastic matching (Fagin and

[*] Corresponding author.
E-mail address: fudos@cs.uoi.gr (I. Fudos).

Stockmeyer, 1998; Del Bimbo and Pala, 1997), which requires a significant amount of work per shape and some derived starting points as a matching guide. QBIC also supports video queries.

Ankerst et al. (1998) present a pixel-based shape similarity retrieval method that allows only minor rotation and translation. Their similarity criterion assumes a very high dimension (linear to the number of pixels in the image), therefore dimensionality reduction is performed.

Mehrotra and Gary (1995) and Gary and Mehrotra (1995, 1993) present a shape-based method, which stores each shape multiple times. More specifically, the shape is positioned by normalizing each of its edges. The space requirements of this method impose a significant overhead. The method is quite susceptible to noise, thus the authors present a sophisticated preprocessing phase to eliminate the noise effects. Finally, the method favors those shapes of the shape base, which have almost the same number of vertices as the query shape.

Hierarchical chamfer matching (see Borgefors, 1988) for hierarchical chamfer matching and (Barrow et al., 1977; Borgefors, 1984) for chamfer matching) creates a distance image using information from the edges, and then tries to minimize the root mean square average of the values in the distance map that a contour hit. Hierarchical chamfer matching gives quite accurate results rather insensitive to random noise, but involves lengthy computations on every extracted contour per query. In the hierarchical version a resolution pyramid is used to improve the performance of the matching algorithm.

Cohen and Guibas (1997) present an image retrieval method based on geometric hashing. This method calculates the hash signature of the shape based on the contributing line segments. The method has been applied to retrieve Chinese characters and is sensitive to rotation and translation. The geometric hashing described in (Cohen and Guibas, 1997) is not related to the geometric hashing presented in Section 3.6.

In this work, we present a shape-based method, where information regarding the boundary of objects is automatically extracted and organized to support a poly-logarithmic (in the number of shape vertices) algorithm based on a novel similarity criterion. Specifically, this paper makes the following technical contributions:

- introduces a new similarity criterion for shapes, which works better in the context of image retrieval than traditional similarity criteria;
- describes a novel way of storing shapes which is tolerant to distortion;
- presents an efficient algorithm for finding the closest match to a given query shape; its time complexity is poly-logarithmic in the number of vertices of the shape base assuming uniform distribution of the vertices in the locus of their possible locations. Introduces a new geometric hashing technique compatible with our matching algorithm.

The algorithm can be easily extended to retrieve the k best matches instead of the single best match.

The rest of this paper is organized as follows. Section 2 presents our similarity criterion for shapes and compares it with existing criteria. Section 3 describes the organization of the data describing the shapes, and the algorithm to retrieve the best match of a query shape. Section 4 presents some experimental results, while Section 5 concludes the paper and discusses future work.

2. Similarity criteria

The Hausdorff distance is a well studied similarity measure between two point sets A and B. The directed Hausdorff distance h and the Hausdorff distance H are defined as follows:

$$h(A,B) = \max_{a\in A} \min_{b\in B} d(a,b), \tag{1}$$

$$H(A,B) = \max(h(A,B), h(B,A)), \tag{2}$$

where d is a point-wise measure, such as, the Euclidean distance. An inherent problem with the Hausdorff distance is that a point in A that is farthest from any point in B dominates the distance. To overcome this problem, Huttenlocher and Rucklidge have defined a *generalized discrete Hausdorff distance* (see, e.g., Huttenlocher and Rucklidge, 1992), given by the kth largest distance rather than the maximum

$$h_k(A,B) = k\text{th}_{a\in A} \min_{b\in B} d(a,b), \tag{3}$$

$$H_k(A,B) = \max(h(A,B), h(B,A)). \tag{4}$$

This metric eliminates somehow the farthest-point domination disadvantage of the Hausdorff metric, but works only for a finite set of points (it is mainly used for $k = m/2$, where m is the size of the point set). The generalized Hausdorff distance does not obey the metric properties.

An interesting alternative measure, called *non-linear elastic matching*, is presented in (Fagin and Stockmeyer, 1998). This measure does not obey the traditional metric properties but a relaxed set of metric properties instead. In practice, this provides the same advantages as any metric, and therefore can be used for clustering. However, the arbitrary number of points distributed on the edges, the need of determining certain starting matching points and the complexity of computing such a match (O(mn) using dynamic programming (Arkin et al., 1997)) makes this measure inappropriate for very large data sets.

In our algorithm we use a new similarity criterion based on the average of minimum point distances, which is derived by adapting (1) as follows:

$$h_{\text{avg}}(A,B) = \text{average}_{a\in A} \min_{b\in B} d(a,b). \tag{5}$$

This measure behaves nicely (gives intuitive results) and it can be computed quite efficiently, as is shown in the next section. The metric properties do not hold for this measure either, but in some sense they hold for a representative average set of points probably different from the original point set. Fig. 1 illustrates an example where, using Hausdorff distance, the shape Q is matched with A instead of B (B is intuitively the closest match). According to the similarity measure used in our work, B is indeed closer to Q than A, because the small peak of B is alleviated by the rest of B which is indeed very close to Q.

3. Efficient retrieval of similar shapes

The algorithm is based on two key ideas
- to efficiently handle the noise effects and to ensure rotation, translation and scale invariance we normalize a shape about its diameter and
- to efficiently implement the matching based on the average distance measure (5) presented in Section 2 we use the notion of the ϵ-envelope.

Normalizing about the diameter. In order to match a query shape to the shapes in the database, some kind of "normalization" is applied so that the matching is translation-, rotation- and scaling-independent. In (Gary and Mehrotra, 1995), Mehrotra and Gary normalize each shape about each of its edges: they translate, rotate, and scale the shape so that the edge is positioned at $((0,0),(1,0))$. Although this approach gives good results in many cases, it would fail to retrieve the distorted shape on the right of Fig. 2, if the shape on the left of the figure was used as the query shape. In our retrieval system, instead of normalizing about the edges, we normalize about the diameter of the shape, i.e., by translating, rotating, and scaling so that the pair of shape vertices that are farthest apart are positioned at $(0,0)$ and $(1,0)$. This ensures better results, because the diameter is less susceptible to local distortion (like the one shown in Fig. 2), which is very common in shapes extracted via automated image processing techniques.

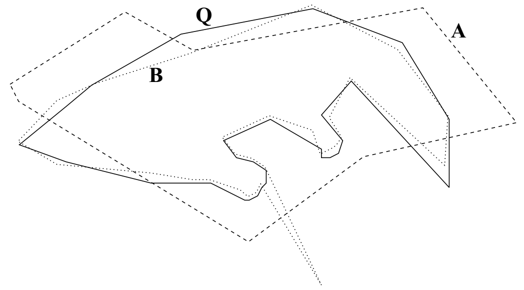


Fig. 1. Depending on the similarity criterion, the query shape Q may be matched with A or B.

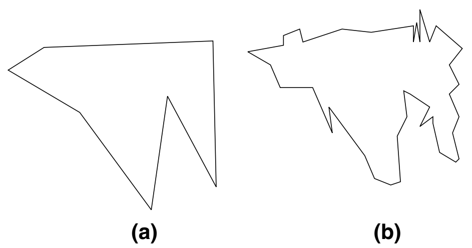


Fig. 2. (a) The query shape; (b) a distorted shape extracted from an image.

The ϵ-envelope. The algorithm works by considering a "fattened" version of the query shape which is computed by taking lines parallel to the query shape edges at some distance ϵ on either side; we call this fattened shape the *ϵ-envelope*. The good matches are expected to fall inside or at least have most of their vertices inside the ϵ-envelope even for small ϵ. Therefore, if we start by using a small initial value of ϵ and keep increasing it, we expect to collect the good matches after a few iterations of this procedure.

The ϵ-envelope can be seen as a collection of trapezoids of height 2ϵ, one for each edge of the query shape. (For simplicity, we assume that ϵ is such that no two trapezoids are overlapping; the method can be extended to handle overlapping trapezoids.)

3.1. Populating the shape database

Populating the database of shapes is done by processing each available shape, a polygon or polyline extracted from an image, as follows. First, we compute the diameter of the shape, i.e., the pair of vertices that exhibit the longest Euclidean distance. In order to achieve even better tolerance to distortion, we will not simply normalize the shape about its diameter, as we alluded earlier; instead, we will normalize it about all its *α-diameters*, i.e., all pairs of vertices whose distance is at least $1-\alpha$ times the length of the diameter ($0 \leqslant \alpha < 1$). For each α-diameter, we scale, rotate, and translate the shape so that the α-diameter is positioned at $((0,0),\ (1,0))$; each shape is stored twice for each α-diameter by taking both ways to match the two vertices defining the α-diameter to the points $(0,0)$ and $(1,0)$. All these "normalized" copies of the shape constitute the *shape base*, the database of shapes.

Of course, a shape with s vertices may have $\Omega(n)$ α-diameters, which would effectively result in an $O(n^2)$-size database to store shapes of $O(n)$ total size. However, this happens to fairly regular shapes; shapes extracted via automated image processing techniques are unlikely to be regular. In fact, experiments have indicated that for $\alpha = 0.15$ the number of copies of each shape is about 12 on the average, including the doubling due to the double storage of a shape for a given α-diameter (the average number of edges per shape in the test set was 20).

3.2. Outline of the matching algorithm

The algorithm works by considering ϵ-envelopes of the query shape for (appropriately) increasing values of ϵ; for each such ϵ, the polygons that have most of their vertices inside the ϵ-envelope are determined and for each of them the value of the similarity measure to the query shape is computed. The algorithm stops whenever the best match has been found, or ϵ has grown "too large" implying that no good matches exist in the shape base. In the latter case, we revert to an alternative but compatible geometric hashing method which we outline in Section 3.6.

In more detail, the basic steps of the algorithm for the retrieval of the database shape that best matches the query shape are

1. We compute an initial value $\epsilon_1 = \epsilon_s$ such that the ϵ_1-envelope is likely to contain at least one shape of the shape base (see Section 3.3). We set $\epsilon_0 = 0$ and we signal that we are in the first iteration by assigning $i = 1$.
2. We collect the vertices of the database shapes that fall in the difference (ϵ_i-envelope − ϵ_{i-1}-envelope); this can be achieved by partitioning this difference into triangles and preprocessing the vertices so that inclusion in a query triangle can be answered fast (simplex range searching). (If no vertices are found then the difference $\epsilon_i - \epsilon_{i-1}$ is increased geometrically.) Additionally, each time we find that a vertex of some shape is inside the above envelope difference, we increase a counter associated with that shape that holds the number of its vertices that are inside the ϵ_i-envelope.
3. If no shape of the shape base has at least a fraction $1-\beta$ of its vertices inside the ϵ_i-envelope (for a parameter β such that $0 \leqslant \beta < 1$), a new larger ϵ is computed (Section 3.5) and we go to step 5.
4. If there are shapes of the shape base that have at least a fraction $1-\beta$ of their vertices inside the ϵ_i-envelope (these are the *candidate shapes*), we process them as described in Section 3.4.

During the processing, we may either conclude that the best match has been found, in which case it is reported to the user and the execution is complete, or a new larger value of ϵ is computed.

5. We increment i and set $\epsilon_i = \epsilon$. If ϵ_i does not exceed $(A/2pl_Q)\log^3 n$, we go to step 2 and repeat the procedure (A is the area of the locus of the normalized shapes (Section 3.3), p is the number of shapes in the shape base, n is the total number of vertices of the p shapes, and l_Q is the length of the perimeter of Q); otherwise, we report the best match so far (if any) and exit. If no match has been found, we employ geometric hashing. The method converges and if there exist similar shapes it retrieves the best match.

3.3. Computing the initial width $2\epsilon_s$ of the ϵ-envelope

We first compute an initial estimate $\hat{\epsilon}$ of the width of the ϵ-envelope based on an estimate $\tilde{K}_{\hat{\epsilon}}$ of the number of vertices that fall inside the envelope. Then, we calculate the actual number $K_{\hat{\epsilon}}$ of vertices of the database shapes. If $K_{\hat{\epsilon}}$ is at least half and no more than twice $\tilde{K}_{\hat{\epsilon}}$, we set $\epsilon_s = \hat{\epsilon}$; otherwise, we adjust ϵ by performing binary search in the values of K_ϵ.

The computation of $\hat{\epsilon}$ is done as follows. Let us compute the area A of the locus of the vertices of the normalized shapes. If the shapes were normalized with respect to their diameter only, then all the vertices would fall in the lune defined by two circles of radius 1 whose centers are at distance 1 apart. Since we store copies of each shape normalized for all pairs of vertices whose distance is at least $1-\alpha$ times the length of the diameter ($0 \leqslant \alpha < 1$), the area A is equal to the area of the lune defined by two circles of radius $1/1-\alpha$ whose centers are at distance 1 apart. This implies that

$$A = \frac{2}{(1-\alpha)^2}\cos^{-1}\left(\frac{1-\alpha}{2}\right) - \sqrt{\frac{1}{(1-\alpha)^2} - \frac{1}{4}}.$$

By assuming uniform distribution of the vertices inside this lune, the average number of vertices inside an ϵ-envelope around the query shape Q is estimated to

$$\bar{K}_\epsilon = \frac{2\epsilon l_Q}{A} n, \tag{6}$$

where l_Q and n are the length of the perimeter of the query shape Q and the total number of vertices of all the shapes of the shape base, respectively.

In order that the initial $\hat{\epsilon}$-envelope contains at least enough vertices for a candidate shape (at least a fraction $1-\beta$ of its vertices lie inside the envelope), we derive that

$$\bar{K}_{\hat{\epsilon}} \geqslant (1-\beta)\frac{n}{p}, \tag{7}$$

where p is the total number of shapes in the shape base.

Estimate (7) may yield the necessary number of vertices, but the probability that all of them belong to the same shape is very small. So, we determine experimentally a $\gamma(n)$, and set $\tilde{K}_{\hat{\epsilon}} = (1-\beta)n\gamma(n)/p$ which implies that $\bar{K}_{\hat{\epsilon}} = \tilde{K}_{\hat{\epsilon}} = (1-\beta)n\gamma(n)/p \Rightarrow$

$$\hat{\epsilon} = \frac{(1-\beta)A}{2pl_Q}\gamma(n). \tag{8}$$

Through experimentation, we have determined that a good choice for $\gamma(n)$ for relatively small shape bases (see Section 4) is: $\gamma(n) = 5\log n$.

3.4. Processing the shapes

We first process all the new candidate shapes, that is, the shapes that have more than a fraction $1-\beta$ of their vertices inside the current ϵ-envelope but did not do so in the previous envelopes. For each such shape P_j, we compute the value of the similarity criterion of P_j with respect to the query shape Q, which we will call the *cost* c_j of P_j: $c_j = \text{average}_{a\in P_j} \min_{b\in Q} d(a,b) \Rightarrow$

$$c_j = \frac{\int_{a\in P_j} \min_{b\in Q} d(a,b)}{\text{length}(P_j)}, \tag{9}$$

where by length(P) we denote the length of the perimeter of P. The computation is done by intersecting each edge of P_j with the Voronoi diagram of Q; the boundary of P_j is thus split into segments that are close to either a vertex or an edge of Q. The contribution of each such segment s in $\int_{a\in s} \min_{b\in Q} d(a,b)$ can then be easily computed

- if s is in the Voronoi region of an edge e of Q and s does not cross e, then the contribution of s is equal to $c(s) = ((d_1 + d_2)/2)\text{length}(s)$, where d_1 and d_2 are the distances of the endpoints of s from e;
- if s is in the Voronoi region of an edge e of Q and s crosses e, then the contribution of s is equal to

  $$c(s) = \frac{d_1^2 + d_2^2}{2(d_1 + d_2)}\text{length}(s),$$

  where d_1 and d_2 are the distances of the endpoints of s from e;
- if s is in the Voronoi region of a vertex v of Q, then the contribution $c(s)$ is given by a more complicated expression, which (of course) only depends on the coordinates of the endpoints of s and of the vertex v.

Then,

$$c_j = \frac{\sum_s c(s)}{\text{length}(P_j)}. \tag{10}$$

If c_j is less than the cost $c_{\max}$ of the best match so far, then P_j becomes the current best match and $c_{\max}$ is set equal to c_j.

Next, we process all the shapes that are not yet candidates (and have at least a vertex other than $(0,0)$ and $(1,0)$ in the current ϵ-envelope), in order to determine whether we have found the best matches, and if not to produce a new larger ϵ for the ϵ-envelope. So, for each of these shapes, say, S_j, we compute the contribution $\int_{a\in e} \min_{b\in Q} d(a,b)$ of each of its edges e that has at least one endpoint inside the ϵ-envelope. Let the sum of all these contributions be t_j and let the total length of all these edges be d_j. We check whether

$$\frac{1}{\text{length}(S_j)}(t_j + \tfrac{\epsilon}{2}(\text{length}(S_j) - d_j)) > c_{\max}.$$

If yes, the cost of S_j will not be less than $c_{\max}$; this is so, because the edges of S_j with at least one endpoint in the current ϵ-envelope contribute t_j in $\int_{a\in e} \min_{b\in Q} d(a,b)$, whereas the remaining edges will contribute more than $\epsilon/2$ times their length (each edge has both endpoints at distance larger than ϵ away from Q). So, if the above inequality holds, we ignore S_j from now on. Otherwise, we compute

$$\epsilon_j = \frac{2(\text{length}(S_j)c_{\max} - t_j)}{\text{length}(S_j) - d_j}$$

which turns the previous inequality into equality. Note that ϵ_j is larger than the current width ϵ of the envelope.

After all the S_j's have been processed, we consider the set of collected ϵ_j's. If the set is empty, then we have found the best match and we stop. Otherwise, we select the smallest element of the set and we use it as the new width ϵ of the envelope.

3.5. Increasing ϵ when there are no candidate shapes

In this case, all the shapes of our shape base have less than a fraction $1-\beta$ of their vertices inside the current ϵ_i-envelope. Then for each of the shapes that have a vertex other than $(0,0)$ and $(1,0)$ in the envelope, we do the following. Let P_j be such a shape with n_j vertices and let V_j be the set of its vertices that are inside the envelope. Consider the set V_j' of vertices of P_j that are adjacent to vertices in V_j; for each vertex v_k in $V_j' - V_j$, we compute the shortest distance $d_Q(v_k)$ of v_k from the query shape Q. If the total number of vertices in $V_j \cup V_j'$ exceeds $(1-\beta)n_j$ (i.e., there are more than a fraction $1-\beta$ of P_j's vertices in $V_j \cup V_j'$), then we set ϵ_j' equal to the $((1-\beta)n_j - |V_j|)$th smallest distance $d_Q(v_k)$; this implies that P_j will be a candidate shape in the ϵ_j'-envelope. In the case that the total number of vertices in $V_j \cup V_j'$ does not exceed $(1-\beta)n_j$, then we set ϵ_j' equal to $((1-\beta)n_j/|V_j|)\epsilon$ (i.e., we use linear interpolation in order to estimate the width of the envelope for which P_j will be a candidate shape).

After all these ϵ_j's have been computed, we collect the smallest among them and use it as the new ϵ of the envelope.

3.6. Geometric hashing

When a match cannot be found and the envelope is large enough, suggesting that the matching algorithm has become inefficient, we revert to the *geometric hashing* method outlined in this section.

This method uses a finite family of curves which cover uniformly the locus of the vertices of the shapes normalized about their diameter, i.e., the

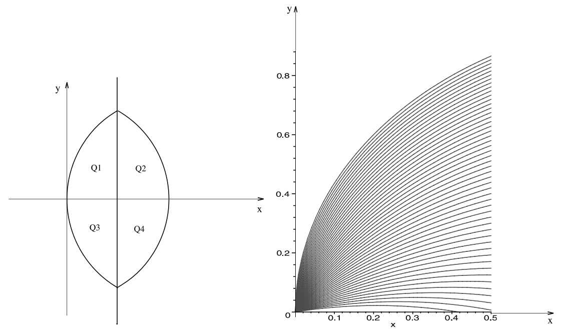


Fig. 3. (Left) the four quarters of the lune; (right) hash curves for the upper left quarter.

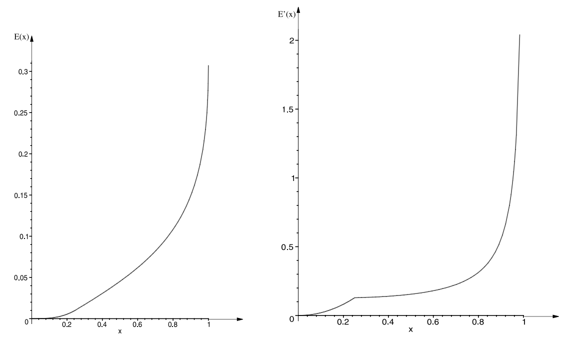


Fig. 4. (Left) graph of $E(x)$; (right) graph of $\vartheta E(x)/\vartheta x$.

lune defined by two circles with radius 1 centered at $(0,0)$ and $(1,0)$. Because the shapes in the shape base have been normalized about their α-diameters, some vertices fall outside this lune; these vertices are treated as if they are located on the boundary of the lune for the purpose of this section.

We consider the partition of the lune in the four quarters Q_1, Q_2, Q_3 and Q_4 as is illustrated in Fig. 3 (left). The uniform coverage of each quarter is achieved by requiring equal areas between consecutive curves of the family. We have considered different families of conic curves, trying to increase the retrieval accuracy, while minimizing the computational complexity of finding the curve closest to a given shape. An interesting family is the family of circles of constant radius 1 which pass through $(0,0)$ for Q_1 and Q_3 and through $(1,0)$ for Q_2 and Q_4. In particular, to partition the upper left quarter Q_1 into k regions of equal area, we use k cyclic arcs that belong to circles whose centers lie on a circle of radius 1 centered at $(0,0)$; the ith arc is derived from a circle of radius 1 with center $(x_{i,} - \sqrt{1-x_i^2})$, where x_i is obtained for each $i = 1,\dots,k$ by solving the following equation for $x = x_i$: $E(x) = \int_0^{\min(2x,1/2)} \left(\sqrt{1-(t-x)^2} - \sqrt{1-x^2}\right) \mathrm{d}t \Rightarrow$

$$E(x) = \frac{A_0}{4}\frac{i}{k}, \tag{11}$$

where A_0 is the area of the lune. It turns out that $E(x)$ and $\theta E/\theta x$ are both continuous in $[0,1]$ (Fig. 4). Thus, fast gradient-based numerical methods can be used to determine x_i from the above equation. For $k = 50$, Fig. 3 (right) illustrates the 50 arcs that were derived by means of the above method.

The hashing works as follows: For each shape of our shape base, we partition its vertices in four sets depending on whether they fall in $Q_1,\dots,Q_4$. Then, for each such set, we compute the hash curve C of the corresponding quarter that minimizes the average distance of these vertices from C. The average distance exhibits only one local minimum (which is also the only local extreme) in the continuous space of curves. To find the closest curve segment we may either perform a binary search in the discrete space of curves that partition our space or we may find the minimum in the continuous space by using a numerical method and then select the discrete neighbor that lies closest to our polygon. The above procedure applied to each of the four quarters results in having each shape associated with one hash curve in each of Q_1, Q_2, Q_3 and Q_4.

By increasing the number of curves, we are able to have a small, on the average, number of shapes associated with each hash curve. Shapes that are close to each other will be associated with the same or neighboring curves; neighboring curves may however be associated with dissimilar shapes.

To retrieve a good match for a given query shape, we apply the hashing procedure to the query shape: we partition its vertices, compute the corresponding curves in each of Q_1, Q_2, Q_3, Q_4, and collect the shapes of the shape base associated with these curves. Finally, we apply the similarity measure between each of the collected shapes and our query shape and report the shape that exhibits the lowest cost. Given that we expect to have a constant number of associated shapes per hash curve, finding the closest match takes time logarithmic in the number of curves in the family.

3.7. Time complexity of the matching algorithm

Before analyzing the time complexities of each of the steps of the algorithm, we recall that in order to compute the similarity measure, we make use of the Voronoi diagram of the query shape Q. This can be computed in $O(m \log m)$ time, where m is the size of Q.

Step 1 of the algorithm begins with the computation of $\hat{\epsilon}$ which takes $O(1)$ time. Then, the number of vertices that fall inside the $\hat{\epsilon}$-envelope is computed; this can be done in $O(\text{poly-}\log n)$ time using simplex range counting algorithms and quadratic or near-quadratic space data structures. If the computed number greatly differs from the expected number, $O(\log n)$ repetitions of the previous computation are done, resulting in $O(\text{poly-}\log n)$ total time for this step.

In step 2, we need to compute the vertices of the shapes in our database that fall in the difference of the ϵ_i-envelope – ϵ_{i-1}-envelope (this ensures that a vertex will not be processed or counted multiple times). The difference of the m trapezoids (one for each of the m edges of the query shape) can be decomposed into $O(m)$ triangles which can be used with simplex range reporting data structures of near-quadratic space complexity that take $O(\log^3 n + \kappa)$ time per query triangle, where n is the total number of vertices of the shape base and κ is the number of vertices that fall in the triangle (Goodman and O'Rourke, 1997). (There are also quadratic-size data structures that allow for $O(\log n + \kappa)$ query time by employing fractional cascading, Chazelle and Guibas, 1986.) Thus completing the ith iteration of step 2 takes $O(m \log^3 n + K_i)$ time in total, where K_i is the number of vertices in all the query triangles for that iteration.

The ith iteration of step 3 takes $O(mK_i)$ time, where K_i is again the number of vertices between the ϵ_i-envelope and the ϵ_{i-1}-envelope.

Step 4 involves processing the new candidate shapes and the non-candidate shapes. The former takes time $O(m|P_i|)$, where $|P_i|$ denotes the number of vertices of P_i. Processing the non-candidate shapes can be performed in $O(1 + mK_i)$ time, by maintaining the contributions of vertices in previous envelopes and simply adding the contributions of vertices between the ϵ_i-envelope and the ϵ_{i-1}-envelope. Step 5 takes constant time.

The overall time complexity after r iterations is therefore

$$\begin{aligned} &O(m \log m) + O(\text{poly-}\log n) \\ &\quad + \sum_{i=1}^{r} O(\text{poly-}\log n + mK_i) \\ &\quad = O(m \log m) + O(r\ \text{poly-}\log n) + O(mK), \end{aligned}$$

where K is the total number of vertices processed and since the total number of candidate shapes is $O(K)$. This is $O(r\ \text{poly-}\log n + K)$ since the size m of the query shape is constant. Finally, by assuming uniform distribution of the vertices inside the lune and in light of the test for ϵ_i in step 5, the number K of vertices and the number r of iterations is expected to be poly-logarithmic in n, and therefore the total time complexity is poly-logarithmic in n.

4. A prototype system for shape-based image retrieval

We have developed a prototype interactive system that implements the proposed geometric-similarity approach. The system uses external storage for the shape base and the auxiliary data structures. The geometric-similarity algorithm has been implemented in *C* and the user interface has been developed using Tcl/Tk. We currently have a stable version running on a Sun Solaris platform. The software is easily portable to other platforms as well.

GeoSIR provides also utilities for edge extraction and detection of object boundaries based on the ipp software (Nelson) and a cluster decomposition algorithm that we have developed. When adding a new image in the image base we process the image and extract shapes that describe sufficiently the boundary of each object. These shapes are non-self-intersecting polylines either open or closed. We first perform image processing that achieves segment approximation of boundaries. We then detect clusters of polylines that describe the boundary of objects. Each such cluster consists of one or more non-self-intersecting polylines that share edges or vertices.

In Fig. 5(a) we see an image and in Fig. 5(b) we see the same image after it has been processed for segment detection. The result of the cluster detection is shown in Fig. 6. There are seven clusters of shapes (A–G). The shapes are approximated by adequately small line segments, connected in polylines. Several heuristics may be used to minimize noise. Our method has been designed to be tolerant to such noise situations. Thus, noise elimination is important only for reducing the effective size of the shape base.

After the polyline clusters have been determined, each cluster is decomposed in a number of non-self-intersecting polylines. There are several different decompositions; achieving a good decomposition is an important issue, but we are not treating it here. During cluster detection and decomposition, certain relations are recorded concerning the relative size and positioning of these shapes with respect to other shapes of the same or different clusters. The efficient use of this information (see, e.g., Nabil et al., 1996) is an important direction for future research.

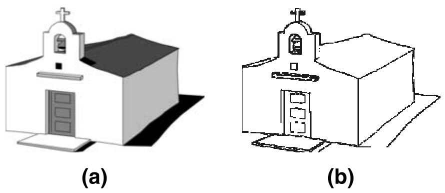


Fig. 5. (a) The original image (converted in grey scale) and (b) after the edge extraction.

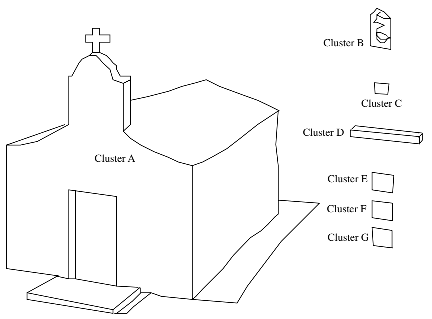


Fig. 6. Seven clusters of shapes (A–G) have been detected.

4.1. User dialogue

The overall user dialogue is illustrated in Fig. 7. The user is first presented with a workspace where she/he can draft a query sketch. This sketch is then decomposed in non-self-intersecting polylines. Initially, the system attempts to use the incremental "fattening" algorithm to find the best match(es). If it fails to find a close match, geometric hashing is used for approximate retrieval. If the user is not satisfied by the returned result(s), she/he can edit the query sketch, specify certain polylines (open or closed) that are of special interest and re-apply the retrieval process. Fig. 9 depicts a snapshot of the GUI.

4.2. Experiments

We have performed experiments with around 100 images from an animation database (like the chapel of Fig. 5) and 350 actual shapes. Each shape is stored on the average approximately 12

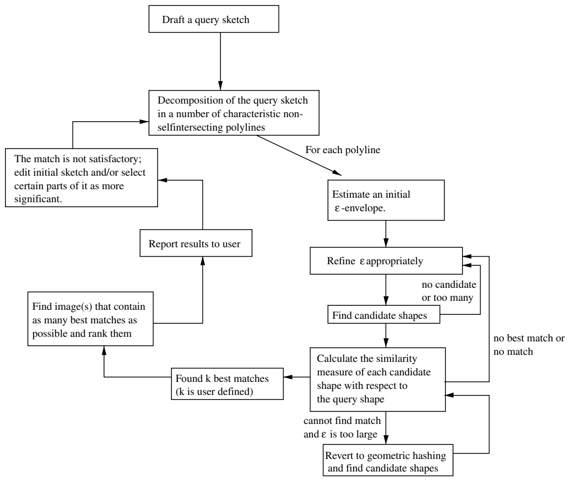

Draft a query sketch
Decomposition of the query sketch in a number of characteristic non-selfintersecting polylines
For each polyline
The match is not satisfactory; edit initial sketch and/or select certain parts of it as more significant.
Estimate an initial ε-envelope.
Report results to user
Refine ε appropriately
no candidate or too many
Find image(s) that contain as many best matches as possible and rank them
Find candidate shapes
no best match or no match
Calculate the similarity measure of each candidate shape with respect to the query shape
Found k best matches (k is user defined)
cannot find match and ε is too large
Revert to geometric hashing and find candidate shapes

Fig. 7. User interaction.

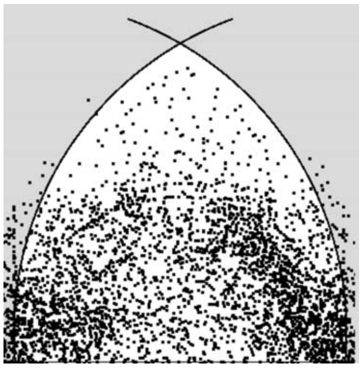
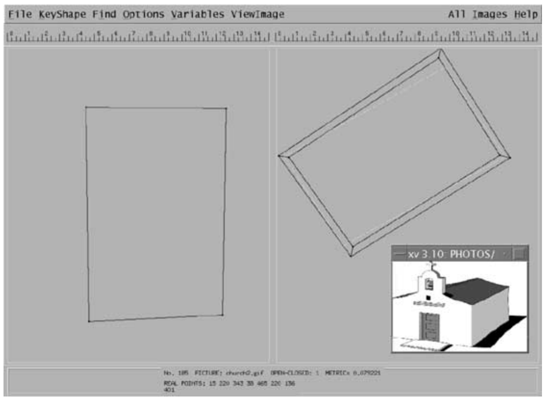


Fig. 8. Distribution of the vertices inside the upper part of the lune; the lower part is symmetric.

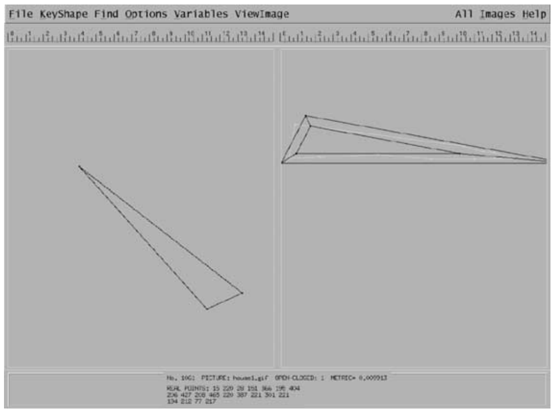


```
No. 105  PICTURE: church2.gif  OPEN-CLOSED: 1  METRIC= 0.079221
REAL POINTS: 15 220 343 38 465 220 136
401
```

Fig. 9. Querying with an orthogonal shape; the door of the church is the best match.

times resulting in a shape base populated with 3000 normalized shapes. The total number of vertices was $n = 30\,000$. The distribution of the vertices was not uniform because of the specialized nature of the images (see Fig. 8).

In the experiments we used $\alpha = 0.15$, $\beta = 0.15$ and $\gamma(n) = 5\log_2 n$. The initial estimation of K_ϵ was usually very close to the actual number of vertices that fell in the envelope. In the case of Fig. 9, we find a best match with the first iteration with cost $c = 0.079221$ since two edges of the matched shapes are partially outside the envelope. Note that all shapes are normalized so that their diameters are have length 1. Thus a cost of this size roughly denotes an average difference of about 8% of the query's actual size between the query and



```
No. 106: PICTURE: house1.gif  OPEN-CLOSED: 1  METRIC= 0.000913
REAL POINTS: 15 220 28 151 366 195 404
206 427 208 465 220 387 221 301 221
134 212 77 217
```

Fig. 10. The best match to this triangular query is a polygon with 10 vertices.

matched shape. Finally, in Fig. 10 we find a best match after a second refinement iteration; in this case, the returned shape for a triangular query shape is a polygon with 10 vertices that has a cost around $c = 0.0008492$, indicating an average distance of about 0.08% of the query's actual size. Even though the vertices are not uniformly distributed in the lune, the algorithm behaves as expected in terms of number of iterations and time complexity.

5. Conclusions and future work

We have presented an efficient noise tolerant shape-based approach to image retrieval. Our system combines two powerful methods, an efficient algorithm for retrieving shapes based on a novel similarity criterion and a geometric hashing technique, to maximize efficiency and intuitiveness. The algorithm currently yields the best match, but it can be easily extended the k best matches; in this case, a heap of size k is used to hold the k current best candidates.

We are currently incorporating this algorithm in a video retrieval system, which will allow us to experiment with larger shape bases. Other future research directions include finding alternative ways to do the range searching (whose space requirement is high), ensuring robust calculations, adding 3D awareness support, and using relative position information for allowing more complicated queries such as containment and tangency.

References

Ankerst, M., Kriegel, H.P., Seidl, T., 1998. Multistep approach for shape similarity search in image databases. IEEE Trans. Knowledge Data Eng. 10 (6), 996–1004.

Arkin, E.M., Chew, L.P., Huttenlocher, D.P., Kedem, K., Mitchell, J.S.B., 1997. An efficiently computable metric for comparing polygonal shapes. IEEE Trans. Knowledge Data Eng. 13 (3), 209–216.

Barrow, H.G., Tenenbaum, J.M., Bolles, R.C., Wolf, H.C., 1977. Parametric correspondence and chamfer matching: Two new techniques for image matching. In: Proc. 5th IJCAI, Cambridge, MA, pp. 659–663.

Borgefors, G., 1984. An improved version of the chamfer matching algorithm. In: ICPR1984, pp. 1175–1177.

Borgefors, G., 1988. Hierarchical chamfer matching: A parametric edge matching algorithm. IEEE Trans. Pattern Anal. Machine Intell. 10 (6), 849–865.

Carson, C., Ogle, V.E., 1996. Storage and retrieval of feature data for a very large online image collection. IEEE Bull. Tech. Comm. Data Eng. 19 (4), 19–27.

Chazelle, B., Guibas, L.J., 1986. Fractional cascading: I. a data structuring technique; II. applications. Algorithmica 1, 133–191.

Cohen, S., Guibas, L., 1997. Shape-based image retrieval using geometric hashing. In: Proc. ARPA Image Understanding Workshop, pp. 669–674.

Del Bimbo, A., Pala, P., 1997. Visual image retrieval by elastic matching of user sketches. IEEE Trans. Knowledge Data Eng. 19 (2), 121–132.

Fagin, R., Stockmeyer, L., 1998. Relaxing the triangle inequality in pattern matching. Internat. J. Comput. Vision 28 (3), 219–231.

Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., Yanker, P., 1995. QBIC: Query by image and video content. IEEE Comput. 28 (9), 23–32.

Gary, J.E., Mehrotra, R., 1993. Similar shape retrieval using a structural feature index. Inf. Systems 18 (7), 527–537.

Gary, J.E., Mehrotra, R., 1995. Feature-index-based similar shape retrieval. In: Spaccapietra, S., Jain, R. (Eds.), Visual Database Systems, Vol. 3, pp. 46–65.

Goodman, J.E., O'Rourke, J., 1997. Handbook of Discrete and Computational Geometry. CRC Press, LLC.

Huttenlocher, D.P., Rucklidge, W.J., 1992. A multi-resolution technique for comparing images using the Hausdorff distance. Technical Report TR92-1321, CS Department, Cornell University.

IBM. IBM's query by image content (QBIC) homepage: http://wwwqbic.almaden.ibm.com.

Mehrotra, R., Gary, J.E., 1995. Similar-shape retrieval in shape data management. IEEE Comput. 28 (9), 57–62.

Nabil, M., Ngu, A.H.H., Shepherd, J., 1996. Picture similarity retrieval using the 2D projection interval representation. IEEE Trans. Knowledge Data Eng. 8 (4), 533–539.

Nelson, R.C., Boundary extraction by lineal feature growing. http://www.cs.rochester.edu/users/faculty/ nelson/research/boundaries/boundaries.html.

Niblack, W., Barber, R., Equitz, W., Flickner, M., Glassman, E., Petkovic, D., Yanker, P., 1993. The QBIC project: querying images by content using color, texture and shape. In: Proc. SPIE Conf. on Storage Retrieval for Image and Video Databases, SPIE, Vol. 1908, pp. 173–181.