An Efficient Shape-Based Approach to Image Retrieval*

Ioannis Fudos and Leonidas Palios

Department of Computer Science, University of Ioannina
GR45110 Ioannina, Greece
{fudos,palios}@cs.uoi.gr

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.

Keywords: image retrieval, shape-based matching, image bases

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 [7]. 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 [10,15], a system developed at IBM Almaden supports retrieval by color histograms, texture samples (based on coarseness, contrast and directionality), and shape. QBIC

* This work was supported in part by a GSRT (General Secretariat of Research and Technology) Bilateral Research Cooperation Grant between Greece and the Czech Republic

G. Borgefors, I. Nyström, and G. Sanniti di Baja (Eds.): DGCI 2000, LNCS 1953, pp. 505–517, 2000.

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 [17], or by clustering using nonlinear elastic matching [9,4], 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 [1] 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.

Gary and Mehrotra [16,12,11] 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 [6] for hierarchical chamfer matching and [3,5] 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.

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;
- 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:
$h(A,B) = \max_{a\in A}\ \min_{b\in B}\ d(a,b),\ \ H(A,B) = \max(h(A,B), h(B,A))$,
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. [14]), given by the k-th largest distance rather than the maximum:
$h_k(A,B) = \mathrm{kth}_{a\in A}\ \min_{b\in B}\ d(a,b),\ \ H_k(A,B) = \max(h(A,B), h(B,A))$.

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 *nonlinear elastic matching*, is presented in [9]. 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 [2]) 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:

$$h_{avg}(A,B) = \mathrm{average}_{a\in A}\ \min_{b\in B}\ d(a,b)$$

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. Figure 1 (left) 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.

3 Efficient Retrieval of Similar Shapes

The algorithm is based on two key ideas: normalizing a shape about its diameter and 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 [12], 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

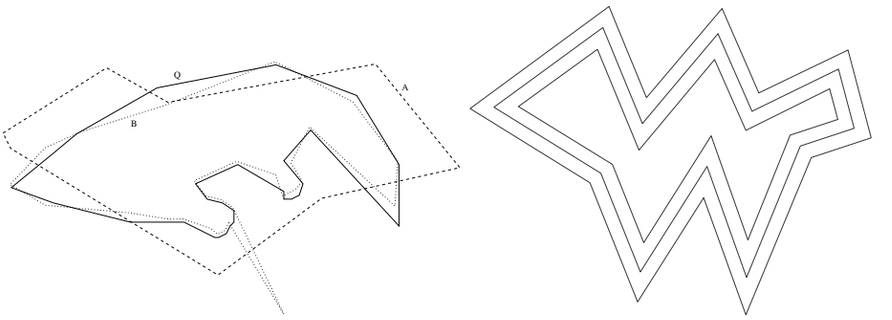

Fig. 1. (left) Depending on the similarity criterion, the query shape Q may be matched with A or B; (right) the ϵ-envelope

approach gives good results in many cases, it would fail to retrieve the distorted shape on the right of Figure 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 Figure 2), which is very common in shapes extracted via automated image processing techniques.

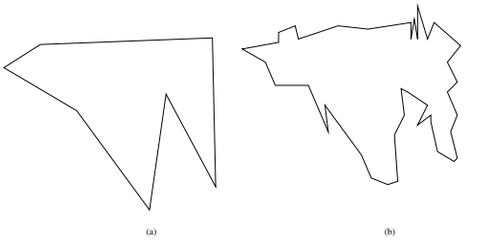

Fig. 2. (a) the query shape; (b) a distorted shape extracted from an image

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 (Figure 1 (right)); 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 ($0 \leq \alpha < 1$). 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 is outlined in the technical report version, due to space limitations.

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 $0 \leq \beta < 1$), 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 $\frac{A}{2\,p\,l_Q}\log^3 n$, 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 ($0 \leq \alpha < 1$), the area A is equal to the area of the lune defined by two circles of radius $\frac{1}{1-\alpha}$ whose centers are at distance 1 apart. This implies that $A = \frac{2}{(1-\alpha)^2}\,cos^{-1}(\frac{1-\alpha}{2}) - \sqrt{\frac{1}{(1-\alpha)^2} - \frac{1}{4}}$.

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:
$\tilde{K}_\epsilon = \frac{2\,\epsilon\,l_Q}{A}n$, 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:
$\tilde{K}_{\hat{\epsilon}} \geq (1-\beta)\frac{n}{p}$ where p is the total number of shapes in the shape base.

The above estimate 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 $\tilde{K}_{\hat{\epsilon}} = (1-\beta)\frac{n}{p}\gamma(n)$ which implies that $\bar{K}_{\hat{\epsilon}} = \tilde{K}_{\hat{\epsilon}} = (1-\beta)\frac{n}{p}\gamma(n) \;\Rightarrow\; \hat{\epsilon} = \frac{(1-\beta)A}{2\,p\,l_Q}\gamma(n)$.

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:

$c_j = \text{average}_{a\in P_j} \min_{b\in Q} d(a,b) = \frac{\int_{a\in P_j} \min_{b\in Q}\, d(a,b)}{length(P_j)}$, 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 $c(s) = \frac{d_1+d_2}{2}\, length(s)$, 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 $c(s) = \frac{d_1^2+d_2^2}{2(d_1+d_2)}\, length(s)$, 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, $c_j = \frac{\sum_s c(s)}{length(P_j)}$. 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 $\frac{1}{length(S_j)}(t_j + \frac{\epsilon}{2}(length(S_j) - d_j)) > c_{max}$. 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, $\epsilon_j = \frac{2(length(S_j)\, c_{max} - t_j)}{length(S_j) - d_j}$ 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_js have been processed, we consider the set of collected ϵ_js. 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 in the Absense of 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.

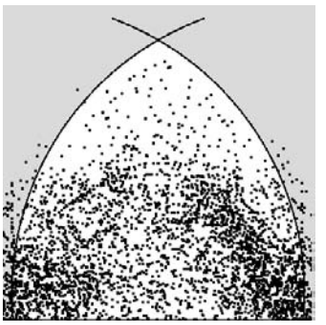

Fig. 3. Distribution of the vertices inside the upper part of the lune; the lower part is symmetric

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 $\big((1-\beta)n_j - |V_j|\big)$-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 $\frac{(1-\beta)n_j}{|V_j|}\epsilon$ (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 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

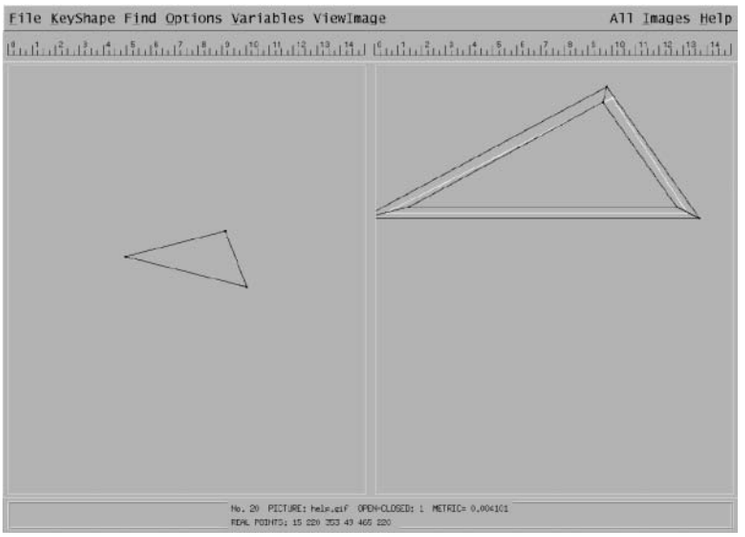

Fig. 4. A best match within the initial guess of the envelope

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 [13]. (There are also quadratic-size data structures that allow for $O(\log n + \kappa)$ query time by employing fractional cascading [8].) Thus completing the i-th 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 i-th 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 ver-

tices 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,
$O(m\ \log m) + O(\text{poly-log}n) + \sum_{i=1}^{r} O(\text{poly-log}n + m\,K_i) =$
$O(m\ \log m) + O(r\,\text{poly-log}n) + O(m\,K)$

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 Experimental Results

The 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 various platforms. The user is first presented with a workspace where she/he can draft a query sketch, which is subsequently presented first to our matching algorithm and in case of failure to the geometric hashing approach.

We have performed experiments with around 100 images and 350 actual shapes. Each shape is stored on the average approximately 12 times resulting in a shape base populated with 3000 normalized shapes. The total number of vertices was $n = 30000$. The distribution of the vertices was not uniform because of the specialized nature of the images (see Figure 3).

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 Figure 4, we started with an initial estimation of 244 for K_ϵ which gave an $\epsilon_1 = 0.0030$ with an actual 180 vertices inside the envelope. In this case we obtain immediately a best match with cost $c = 0.0013$. Similarly in the case of Figure 5, we find a best match with the first iteration with slightly larger cost $c = 0.0792$ since two edges of the matched shapes are partially outside the envelope. Finally, in Figure 6 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 $c = 0.0008$. 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. The algorithm currently yields the best match, but it can be easily

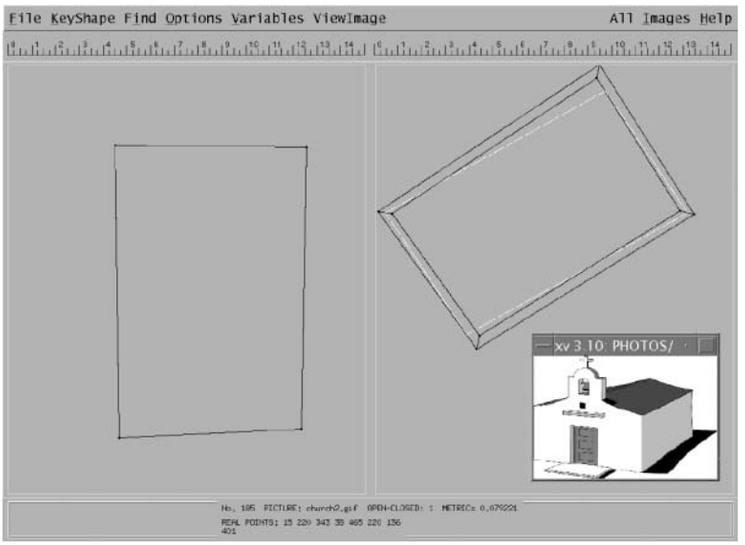

Fig. 5. Querying with an orthogonal shape; the door of the church is the best match

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

1. M. Ankerst, H. P. Kriegel, and T. Seidl. Multistep approach for shape similarity search in image databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):996–1004, 1998. 506
2. E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):209–216, 1997. 507
3. H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proc. of the 5th IJCAI*, pages 659–663, Cambridge, MA, 1977. 506
4. A. Del Bimbo and P. Pala. Visual image retrieval by elastic matching of user sketches. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):121–132, 1997. 506

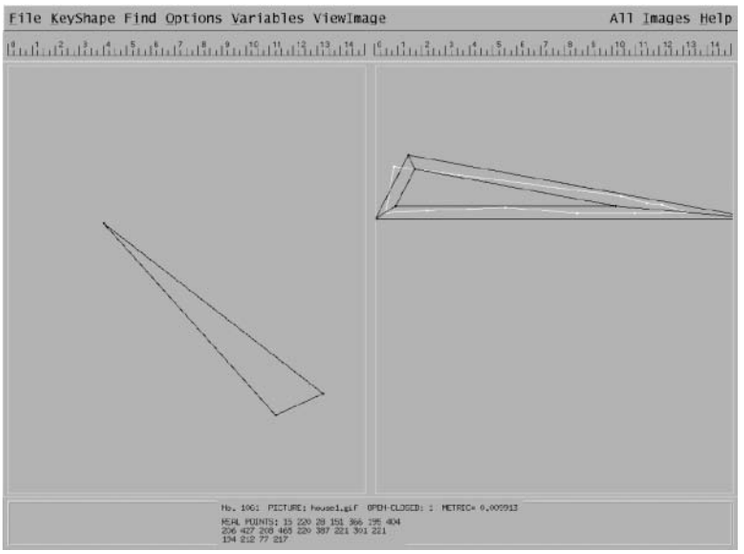

Fig. 6. The best match to this triangular query is a polygon with 10 vertices

5. G. Borgefors. An improved version of the chamfer matching algorithm. In *ICPR1984*, pages 1175–1177, 1984. 506
6. G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, 1988. 506
7. C. Carson and V. E. Ogle. Storage and retrieval of feature data for a very large online image collection. *IEEE Bulletin of the Tech. Comm. on Data Engineering*, 19(4):19–27, 1996. 505
8. B. Chazelle and L. J. Guibas. Fractional cascading: I. a data structuring technique; II. applications. *Algorithmica*, 1:133–191, 1986. 513
9. Ronald Fagin and Larry Stockmeyer. Relaxing the triangle inequality in pattern matching. *International Journal of Computer Vision*, to appear, 1999. 506, 507
10. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. QBIC: Query by image and video content. *IEEE Computer*, 28(9):23–32, 1995. 505
11. J. E. Gary and R. Mehrotra. Similar shape retrieval using a structural feature index. *Information Systems*, 18(7):527–537, 1993. 506
12. J. E. Gary and R. Mehrotra. Feature-index-based similar shape retrieval. In S. Spaccapietra and R. Jain, editors, *Visual Database Systems*, volume 3, pages 46–65, 1995. 506, 507
13. J. E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 1997. 513
14. D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the hausdorff distance. Technical Report TR92-1321, CS Department, Cornell University, 1992. 507
15. IBM. Ibm's query by image content (QBIC) homepage. http: //wwwqbic.almaden.ibm.com. 505

16. R. Mehrotra and J. E. Gary. Similar-shape retrieval in shape data management. *IEEE Computer*, 28(9):57–62, 1995. 506
17. W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glassman, D. Petkovic, and P. Yanker. The QBIC project: querying images by content using color, texture and shape. In *Proc. SPIE Conference on Storage Retrieval for Image and Video Databases*, volume 1908, pages 173–181. SPIE, 1993. 506