Geometric-Similarity Retrieval in Large Image Bases*

Ioannis Fudos, Leonidas Palios and Evaggelia Pitoura
Department of Computer Science, University of Ioannina,
GR45110 Ioannina, Greece
email:{fudos,palios,pitoura}@cs.uoi.gr

Abstract

We propose a novel approach to shape-based image retrieval that builds upon a similarity criterion which is based on the average point set distance. Compared to traditional techniques, such as dimensionality reduction, our method exhibits better behavior in that it maintains the average topology of shapes independently of the number of points used to represent them and is more resilient to noise. An efficient algorithm is presented based on an incremental "fattening" of the query shape until the best match is discovered. The algorithm uses simplex range search techniques and fractional cascading to provide an average poly-logarithmic time complexity on the total number of shape vertices. The algorithm is extended to perform additional fast approximate matching, when there is no image sufficiently similar to the query image. We present techniques for the efficient external storage of the shape base and of the auxiliary geometric data structures used by the algorithm. Finally, we show how our approach can be used for processing queries, containing pairwise relations of object boundaries such as contain, tangent, and overlap. Such queries are either extracted from some user drafted sketch or defined explicitly by the user. Alternative methods are presented for forming query execution plans.

1. Introduction

There is an increasing effort to organize and retrieve images by content based on characteristics such as color, texture and shape. In this paper, we present a novel method for retrieving images similar to a given shape from a large image base. Our method builds upon a shape-similarity criterion which is based on the average point distance between the shapes. Our similarity criterion is tolerant to distortion and is independent of the number or order of vertices in each shape. We provide a poly-logarithmic (log in some small constant power) in the total number of shape vertices algorithm for retrieving shapes similar to a given query shape. The algorithm "fattens" the query shape until the best match is discovered. In the case in which there is no exact match, we provide a complementary geometric hashing approach that returns approximate matches.

We consider and evaluate different methods for external storage of the shape base. We extend our method to support queries that involve multiple shapes and their pairwise positioning. We focus on a restricted set of queries that can be expressed by union, intersection and complement over the result produced by similarity, contain, overlap and disjoint operators. Finally, we report on a prototype implementation of our approach, that incorporates the shape-matching algorithm, the geometric hashing approach and the query processor in an interactive system called GeoSIR.

To the best of our knowledge, this is the first time that the average point distance measure is used for image retrieval. Perhaps, the work most related to ours is the shape-based method proposed by Gary and Mehrotra [21, 16, 15]. In this work, each shape is represented as a vector in multidimensional space and the Euclidean distance is used. Each shape is stored multiple times, once per edge. More specifically, the shape is positioned by normalizing one of its edges. Thus, 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.

A number of methods in the literature perform indexing and retrieval based on global image characteristics such as color, texture, layout, or their combinations. Along these lines, QBIC [13, 19], 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 scal-

[1]Work partially supported by GSRT PAVET Grant Nr 00BE3, and by a University of Ioannina, Research Committee Grant

ing [24], or by clustering using nonlinear elastic matching [12, 6], which requires a significant amount of work per shape and some derived starting points as a matching guide.

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.

Hierarchical chamfer matching (see [9] for hierarchical chamfer matching and [4, 8] for chamfer matching) creates a distance image using information from the edges, and then tries to minimize the sum of the values in the distance map that the contour hit. Hierarchical chamfer matching gives quite accurate results but involves lengthy computations on every extracted contour per query.

Cohen and Guibas [10] 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. However the method is sensitive to rotation and translation. The geometric hashing described in [10] is not related to the geometric hashing presented in our work.

Korn et al [20] present an approach to nearest neighbor search based on the *max morphological distance* [11]. They use n-d R-trees for indexing. Although this distance works well for arbitrary point sets (for examples, tumors and other medical imaging clusters), shape retrieval is again ambiguous due to the high dimensionality of the search space.

Finally, in [5, 7] the problem of arranging the external storage of the images is tackled nicely. Results are presented that are of special interest to parallel similarity search and dynamic environments, where insert and delete operations occur frequently. However, the proposed methods are based on nearest neighbor techniques in high dimensions and cannot be used to handle external storage in our context.

The rest of this paper is organized as follows. Section 2 presents our similarity criterion for shapes, and the shape-similarity matching algorithm. Section 3 introduces geometric hashing for approximate matching. Section 4 focuses on external storage, while Section 5 describes our query processing approach. Section 6 reports on GeoSIR, our prototype system. Section 7 concludes the paper.

2. Geometric Similarity and the Matching Algorithm

2.1. Motivation

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: $h(A,B) = \max_{a\in A} \min_{b\in B} d(a,b)$ and $H(A,B) = \max(h(A,B), h(B,A))$ where d is a pointwise 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. Figure 1 illustrates such an example where, using Hausdorff distance, the shape Q is matched with A instead of B (B is intuitively the closest match).

To overcome this problem, Huttenlocher and Rucklidge have defined a *generalized discrete Hausdorff distance* (see e.g., [18]) given by the k-th largest distance rather than the maximum: $h_k(A,B) = \text{kth}_{a\in A} \min_{b\in B} d(a,b)$ and $H_k(A,B) = \max(h(A,B), h(B,A))$ This metric eliminates somehow the farthest-point domination disadvantage of the Hausdorff metric. However, the method 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 that tries to alleviate the problems of the Hausdorff metric, called *nonlinear elastic matching*, is presented in [12]. This measure does not obey the traditional metric properties but instead a relaxed set of metric properties. 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 ($O(n_A n_B)$ using dynamic programming [3], where n_A and n_B are the number of vertices of shapes A and B respectively) makes this measure inappropriate for very large data sets.

2.2. Geometric Similarity

To eliminate the dominating maximum pair problem, we use a similarity criterion based on the average of minimum point distances:

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

The proposed similarity criterion provides a natural way to match shapes independently of the number or the order of vertices. It clearly does not suffer from the dominating maximum pair of points problem since, in contrast to the Hausdorff metric, it is based on the average point distance. In the example of Figure 1, with our similarity measure, B is closer to Q than A.

This measure can be computed quite efficiently. In addition, it is scale, translation and rotation invariant. 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.

Note that we compute the average over all points of the continuous shape A not just its vertices (in the case of a discrete average on the vertices of A, the median might be

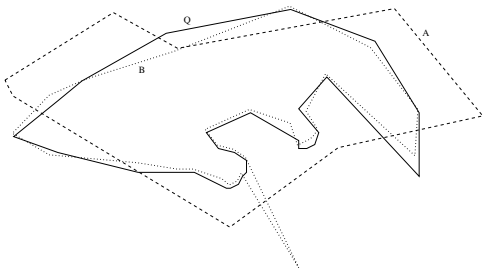

Figure 1. Depending on the similarity criterion, the query shape Q may be matched with A or B.

used instead). Finally, although the max min metrics could be useful in specific applications, for instance to guarantee that there is no point of shape A with distance larger than a threshold from B (e.g., to enforce certain tolerance in matching CAD parts), the average minimum distance metric gives more intuitive results in general similarity matching.

2.3. Efficient Retrieval of Similar Shapes

Before describing our image retrieval algorithm, we present two key ideas of our method: normalizing a shape about its diameter and the notion of the ϵ-envelope.

In order to match a query shape to the shapes in the database, some kind of "normalization" is required so that the matching is translation-, rotation-, and scaling-independent. In [16], 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 $((0,0),(1,0))$. Then, each shape is stored multiple times, twice for each edge. Instead, we normalize about the diameter of the shape. We do so by translating, rotating, and scaling so that the pair of shape vertices that are farthest apart are positioned at $(0,0)$ and $(1,0)$.

Our approach, besides being more space efficient, is less susceptible to local distortion. For example, the Mehrotra and Gary method 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. This is because no pair of edges between the shapes matches. However, our method would match the two shapes. Such distortion is very common in shapes extracted from object boundaries via automated image processing techniques.

Our 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 3); 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.

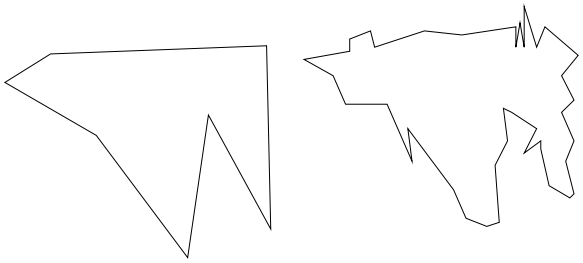

Figure 2. (left) the query shape; (right) a distorted shape extracted from an image.

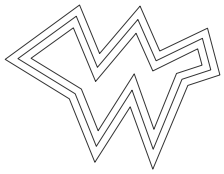

Figure 3. the ϵ-envelope.

2.4. Populating the Shape Database

Shapes are extracted from images and represent object boundaries. We define as shape a non self-intersecting polygon or polyline with no convexity restrictions. Self-intersecting polygons or polylines extracted from an image are decomposed in a number of shapes as described in Section 6. To populate the database of shapes, we process each shape 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 do not simply normalize the shape about its diameter, as we alluded earlier; instead, we 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.

2.5. 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 our similarity measure h_{avg} 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 Section 3.

In more detail, let ϵ_i-envelope be the envelope at iteration i and ϵ_i its width. Initially, for the ϵ_0-envelope, ϵ_0 is 0, meaning that the envelope coincides with the shape. At each consequent iteration of the envelope fattening, we compute which vertices fall inside the difference between the two envelopes (ϵ_i-envelope $-$ ϵ_{i-1}-envelope).

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 ϵ_1 such that the ϵ_1-envelope is likely to contain at least one shape of the shape base. To this end, we iteratively adjust the size of the envelope.

2. We collect the vertices of the database shapes that fall inside 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). 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 envelope is computed 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 to derive bounds on the similarity measure from the corresponding shapes. 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 envelope is computed.

5. We increment i and set ϵ_i to the width of the new larger envelope. 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, 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.

We have proved that the method converges and if there exist similar shapes it retrieves the best match [14]. The choice of the value of constants α and β does not affect the correctness of the algorithm but may improve both the speed of convergence up to some constant and the noise tolerance of the system.

To compute the similarity measure, we use the Voronoi diagram of the query shape Q. This can be computed in $O(m \log m)$ time, where m is the number of vertices of Q.

Step 1 of the algorithm begins with the computation of ϵ_1 which takes $O(1)$ time. Then, the number of vertices that fall inside the ϵ_1-envelope is computed; this can be done in $O(\text{poly-log}n)$[2] time using simplex range counting algorithms and quadratic or near-quadratic space data structures. If no shape falls inside the envelope, then $O(\log n)$ repetitions of the previous computation may be needed to increase the size of the envelope, 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 inside the difference of the ϵ_i-envelope and ϵ_{i-1}-envelope (this ensures that a vertex will not be processed or counted multiple times). The difference of the two envelopes consists of m trapezoids (one for each of the m edges of the query shape), which can be decomposed into $O(m)$ triangles. These triangles 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 inside the triangle [17]. 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 between the ϵ_i-envelope and the ϵ_{i-1}-envelope.

The i-th iteration of step 3 takes $O(mK_i)$ time.

Step 4 involves processing the new candidate shapes (P_i) 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

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

where K is the total number of vertices processed. This is $O(r\ \text{poly-log}n + K)$ since the number m of the vertices of

[2] poly-log$n$, stands for $\log n$ to some constant power ρ: $\log^\rho n$

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. More specifically, we have proved that the expected time complexity is no more that $O(\log^4 n)$. Experimental results however indicate that the actual time complexity is much better.

3. Geometric Hashing

When there is no exact match, we revert to the *geometric hashing* method. Geometric hashing provides us with an approximate match in logarithmic time. The 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 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.

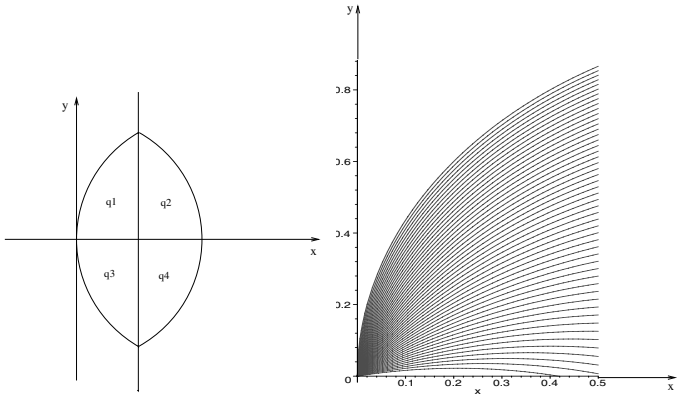

Figure 4. (left) the four quarters of the lune; (right) hash curves for the upper left quarter.

We consider the partition of the lune in the four quarters q_1, q_2, q_3 and q_4 as illustrated in Figure 4 (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 i-th 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 $i = 1, \ldots, k$ by solving the following equation for $x = x_i$:

$$E(x) = \int_0^{\min(2x, \frac{1}{2})} \left(\sqrt{1-(t-x)^2} - \sqrt{1-x^2} \right) dt \; = \; \frac{A_0}{4} \frac{i}{k},$$

where A_0 is the area of the lune. It turns out that $E(x)$ and $\vartheta E / \vartheta x$ are both continuous in $[0,1]$ (Figure 5).

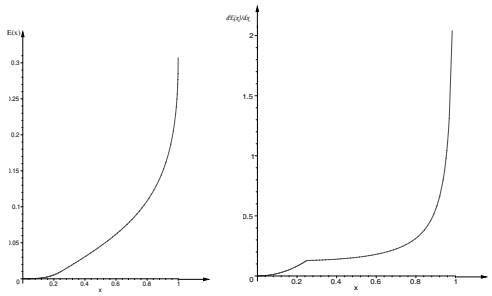

Figure 5. (left) graph of $E(x)$; (right) graph of $\vartheta E(x)/\vartheta x$.

Thus, fast gradient-based numerical methods can be used to determine x_i from the above equation. For $k = 50$, Figure 4 (right) illustrates the 50 arcs that were derived by the above method.

Hashing works as follows: for each shape of the shape base, we partition its vertices in four sets depending on whether they fall in q_1, q_2, q_3, or q_4. Then, for each such set, we compute the hash curve c_i of the corresponding quarter q_i that minimizes the average distance of these vertices from c_i.

Figure 6 depicts the characteristic curve c_1 for a polyline in the first quadrant, which minimizes the average distance of vertices. 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 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

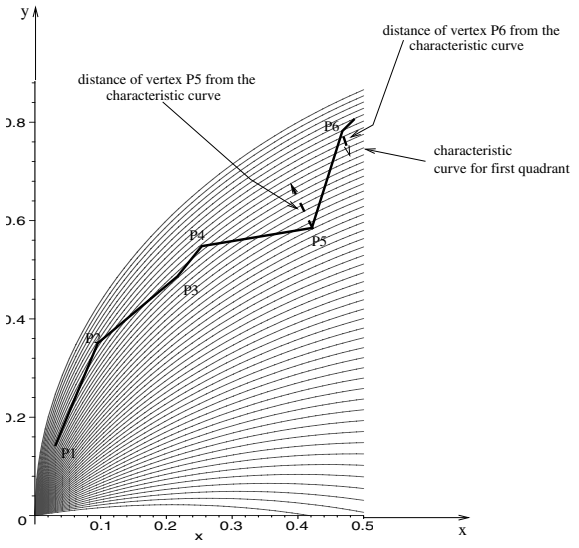

Figure 6. The characteristic curve of this polyline is the one that minimizes the average distance from the vertices. The distances of P_5 and P_6 are illustrated, the other vertices fall almost on the characteristic curve.

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 the query shape and report the one that is closest to the query shape. 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.

4. External Storage

We describe how the image base can be maintained in external storage. First we outline how the shape base and the per shape information is stored in external storage media. Since our image retrieval algorithm preserves locality, in that two shapes which are processed successively are usually similar, our objective is to store similar shapes in adjacent disk locations. To this end, we consider two alternative methods and evaluate their performance.

1. We store the shapes sorted by their characteristic hashing curves.
2. We store the shapes so that to minimize the average similarity measure among shapes stored in adjacent disk locations.

For accommodating the auxiliary data structures in external memory we use optimal range search indexing structures described in [2, 25].

4.1. Sorting by the Characteristic Hashing Curves

Each shape has a quadruple (c_1, c_2, c_3, c_4) of characteristic hash curves, one for each quadrant of the lune. Each characteristic curve is represented by an integer. We consider storing shapes sorted by this quadruple. We have used the following methods of sorting according to the characteristic quadruple of each shape:

i. Sort the shapes by the closest to the mean characteristic curve $c_{mean} = round(\frac{c_1+c_2+c_3+c_4}{4})$

ii. Store the shapes following the lexicographical order of their characteristic quadruples.

iii. Sort the four elements of each quadruple, select the two median curves, and select from the two the one that is closest to the average of the four. Then store the shapes sorted by this element.

We have conducted several experiments to determine the most efficient way of sorting the per shape information on external storage. We have experimented with a database of 10,000 images with an average of 5.5 actual shapes per image. In our test shape base, we have an average number of around 20 vertices per shape. This results in an average size of stored information of around 200 bytes per shape. We then have an average capacity of around 5 shapes per 1Kbyte disk block. In this test set, each shape is stored in average 10 times in our shape base, resulting in around 550,000 shapes, which fit in 110,000 disk blocks. Thus, the actual external storage needed was around 150Mbyte.

In the first experiment, we compare the three methods. We consider an internal memory buffer of size 100k (capable of handling 100 disk blocks). The results are shown in Figure 7. The Figure depicts the mean over a representative experiment set of 15 similarity queries. We have performed the experiment for the best match ($k = 1$), the two best matches ($k = 2$), up to the ten best matches ($k = 10$). Method (i) exhibits the best average time in terms of I/O operations.

In the second experiment, we use internal buffers of various sizes. We have performed the same experiment for $k = 2$ and for a variable size internal memory buffer (1Kbyte - 100Kbyte). Figure 8 depicts the results. The median method (method (iii)) stabilizes faster meaning that locality is preserved better in this method.

The rehashing time is $O(NlogN)$, (assuming a constant number of vertices per shape) where N is the number of shapes in the database. This time is comparable in all three methods, and it is quite I/O intensive.

4.2. Local Optimization of the Average Measure

In this approach, we use our similarity measure for organizing the storage of the shapes in each block. We select

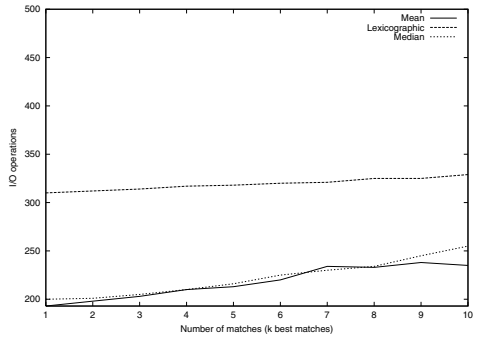

Figure 7. The average number of I/O operations per query for a test set of 15 queries.

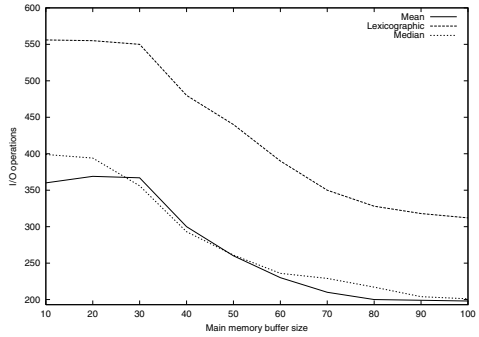

Figure 8. The average number of I/O operations per query for varying buffer size.

the first shape of the first block by a heuristic rule. Then, each subsequent shape in the disk block, is selected so as to minimize the average measure from the previous shapes in the same block. Then for the first shape of the next block we select a shape that minimizes the average distance from the first shapes of the previous five disk blocks. The average number of I/O operations is around 30% better than the best of the previous methods.

Rehashing can be performed in time $O(N^{1.5}logN)$ where N is the number of shapes, if we assume a constant number of vertices per shape. Rehashing is not as I/O intensive as in the previous method.

5. Query Processing

Besides retrieving images that contain a given query shape, our approach supports queries that involve multiple shapes and their pairwise positioning. An example of such a query is: find all images that contain two overlapping shapes similar to query shapes Q_1 and Q_2. We focus on a restricted set of queries that can be expressed by union, intersection and complement over the result produced by similarity, contain, overlap and disjoint operators.

For each image I in the database, we maintain a directed graph $G_I = (V_I, E_I)$ where V_I is the set of shapes in I and E_I is a set of labeled edges (v_1, v_2, $label$), where v_1, $v_2 \in V_I$ and $label \in \{contain, overlap\}$. An edge $v_1 \rightarrow_{overlap} v_2$ (which corresponds to (v_1, v_2, $overlap$)) between two shapes v_1 and v_2 indicates that v_1 $overlaps$ with v_2 and an edge $v_1 \rightarrow_{contain} v_2$ indicates that v_1 $contains$ v_2. There is no edge between shapes that are disjoint. In addition, with each shape S, we maintain an attribute, denoted $S.image$, that identifies the image to which S belongs.

In the following, we denote by DB the set of all images in the database and by DBS the set of all shapes in the database.

5.1. Topological Queries

Let $g\_similar(S_1, S_2)$ be the similarity predicate between two shapes which is true if and only if shape S_1 is similar to shape S_2. Then, we consider a similarity operator $similar(Q)$ that returns all images that contain shapes that are similar to the query shape Q; that is, $similar(Q) = \{I \in DB$: $\exists\, S \in DBS, S.image = I$ and $g\_similar(S, Q)\}$.

Similarly, we define topological predicates, $g\_r(S_1, S_2, \theta)$, refering to the relative position of two shapes S_1 and S_2, where $r \in \{contain, overlap, disjoint\}$ and $\theta \in [-2\pi, 2\pi] \cup \{any\}$. For example, $g\_contain(S_1, S_2, \frac{\pi}{4})$ is true iff S_1 contains S_2 and the signed angle between their diameters is $\frac{\pi}{4}$. In light of these predicates, we define three topological operators that return all images that contain shapes related with the corresponding topological predicate. That is, for a topological operator r, $r(Q_1, Q_2, \theta) = \{I \in DB$: $\exists\, S_1, S_2 \in DBS$, $S_1.image = S_2.image = I$, such that $g\_similar(S_1, Q_1)$ and $g\_similar(S_2, Q_2)$ and $g\_r(S_1, S_2, \theta)\ \}$.

A topogical query is defined as follows:

- The result of a topogical operator and the result of the similarity operator is a topological query.
- If P_1 and P_2 are topological queries, then $P_1 \cap P_2$ is a topological query.
- If P_1 and P_2 are topological queries, then $P_1 \cup P_2$ is a topological query.
- If P_1 is a topological query, then COMPLEMENT(P_1) is a topological query that includes all images not in P_1, that is COMPLEMENT(P_1)= DB - P_1
- If P_1 is a topological query, then (P_1) is a topological query.

For example the topological query:
$similar(Q_1) \cap$ COMPLEMENT($overlap(Q_2, Q_3, any)$)
returns all images in the database that contain a shape similar to Q_1 but do not contain any shape similar to Q_2 that overlaps with a shape similar to Q_3.

5.2. Statistical Estimations

To efficiently implement queries we need an estimate of the size of the sets produced by our operators. Our matching algorithm returns all shapes that are similar to a given query shape Q. Let $shape\_similar(Q) = \{S \in DBS: \ g\_similar(S,Q)\ \}$. The metric $selectivity_{shape\_similar}(Q)$ is an estimate of the size of $shape\_similar(Q)$. Note that if an image contains exactly one of the shapes that are similar to Q, then $selectivity_{shape\_similar}(Q)$ gives an estimate of the size of the query $similar(Q)$.

To this end, we define the number of "significant" vertices ($V_S(Q)$) as:

$$V_S(Q) = \frac{1}{2} \sum_{i=0}^{V(Q)-1} (\pi - \alpha_i)\alpha_i \frac{4}{\pi^2} + \frac{l_{(i-1) \bmod V(Q)} + l_i}{2}$$

where $V(Q)$ is the actual number of vertices of Q, a_i is the positive acute angle that corresponds to the i-th vertex of shape Q, $0 \leq \alpha_i \leq \pi$, and l_i is the legth of the i-th edge (see Figure 9).

$V_S(Q)$ is an estimate of the number of structurally dominating vertices of shape Q, in the sense that $V_S(Q)$ favors vertices that are not degenerate (the corresponding angle is not 0 or π, and the adjacent edges have adequate lengths), and it favors clear cut angles with long adjacent edges. Each vertex contributes a term in $[0,1]$, where 1 is attained when the angle is $\frac{\pi}{2}$ and each adjacent edge has length equal to the diameter of the shape.

For example, in Figure 9(left) we have a normalized shape Q, $\alpha_0 = \frac{\pi}{2}$, $\alpha_1 = \frac{3\pi}{4}$, $\alpha_2 = \frac{\pi}{2}$, $\alpha_3 = \frac{3\pi}{4}$, $\alpha_4 = \frac{\pi}{2}$, $l_0 = l_3 = l_4 = \frac{\sqrt{10}}{5}$, $l_1 = l_2 = \frac{\sqrt{5}}{5}$; thus vertices V_0, V_4 contribute a term $\frac{1}{2} + \frac{\sqrt{10}}{10}$ each, vertices V_1, V_3 contribute a term $\frac{3}{8} + \frac{(2+\sqrt{2})\sqrt{10}}{20}$ and vertex V_2 contributes a term $\frac{1}{2} + \frac{\sqrt{5}}{10}$. In Figure 9(right) we have a normalized shape Q' with 7 vertices, however the significant vertices of Q are the same as in Q'.

Clearly, $0 \leq V_S(Q) \leq V(Q)$.

We have experimentally established that the size of the result of a similar query on Q is inversely proportional to the number of significant vertices $V_S(Q)$.

$$selectivity_{shape\_similar}(Q) = \frac{c}{V_S(Q)}$$

where c is a constant that depends on the size of the shape base and the application domain. This constant is adapted statistically everytime a query is performed.

Figure 10 reports two of our experiments. Both experiments are for the same image domain but for different sizes of the shape base. The size of the shape base for Experiment 1 is twice that for Experiment 2. The experiments validate the hyperbolic behavior of the number of the shape matches in terms of $V_S(Q)$.

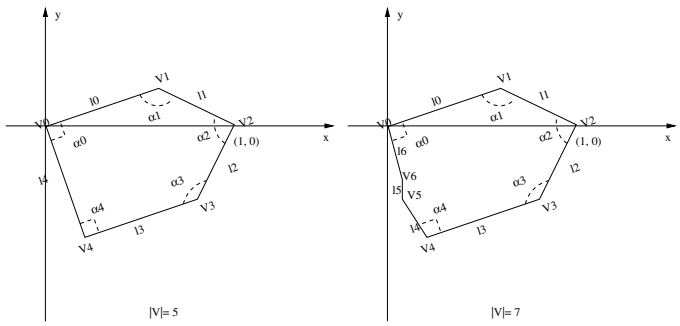

Figure 9. (left) A normalized shape Q with 5 vertices; (right) a normalized shape Q' with 7 vertices. The quantity $V_S(Q)$ is almost equal to $V_S(Q')$.

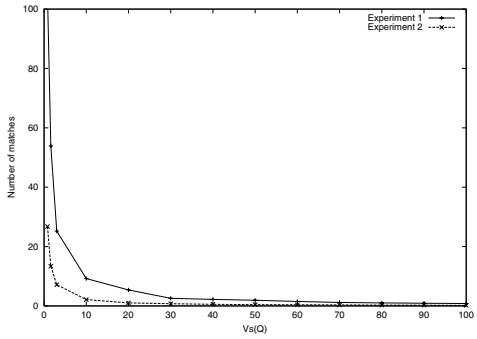

Figure 10. Determining experimentally the number of similar shapes.

5.3. Processing a Single Operator

Our matching algorithm returns all shapes that are similar to a given query shape Q, i.e., the set $shape\_similar(Q)$. To compute the operator $similar(Q)$, we just check the attribute image ($S.image$) for all $S \in shape\_similar(Q)$. This takes time proportional to the size of $shape\_similar(Q)$.

To compute the angle between two shapes S_1 and S_2, we retrieve the inverse normalization transformations T_1 and T_2 from the shape base, apply them on the diameter (which is always the vector $diam = ((0,0),(1,0))$) and calculate the ordered signed angle between $T_1(diam)$ and $T_2(diam)$.

To compute each topological operator $r(Q_1, Q_2, \theta)$ there are two ways.

1. We start by computing first the similarity oper-

ator for the query shape that produces the smallest set. Let this be the query on shape Q_2, that is, let $selectivity_{shape\_similar}(Q_1) \geq selectivity_{shape\_similar}(Q_2)$.

Compute the set $shape\_similar(Q_2)$;
For each $S_2 \in shape\_similar(Q_2)$
  Let $I = S_2.image$ and $G_I = (V_I, E_I)$
    be the corresponding graph;
  For each edge $S_2 \rightarrow_r S_1$ in E_I
    If S_1 is similar to Q_1 and the angle is θ
      Add I to the result;

2. We compute both the set of shapes that are similar to Q_1 and the set of shapes that are similar to Q_2.

Compute $I_1 = shape\_similar(Q_1)$;
Compute $I_2 = shape\_similar(Q_2)$;
Compute $similar(Q_1)$, $similar(Q_2)$ and
  $SI = similar(Q_1) \cap similar(Q_2)$;
    $\{SI$ is the set of all images that contain both a shape similar to Q_1 and a shape similar to Q_2 $\}$
For each $S_1 \in shape\_similar(Q_1)$
  such that S_1.image $\in SI$
    Let $I = S_1.image$ and $G_I = (V_I, E_I)$
      be the corresponding graph;
    For each edge $S_2 \rightarrow_r S_1$ in E_I
      If S_2 in $similar(Q_2)$ and the angle is θ
        Add I to the result;

5.4. Processing Queries

There are several ways to execute a query that contains set operators. We re-write the initial query into the form $t_1 \cup t_2 \cup \ldots \cup t_n$, where each t_i contains only intersection and complement operators.

Let $t_i = p_{i_1} \cap p_{i_2} \cap \ldots \cap p_{i_k}$, where each p_{i_j} is either a (topological or similarity) operator or the complement of an operator. To compute each t_i, we use the estimations for the size of the result of each of the p_{i_j}'s. If p_{i_j} is the similarity operator, i.e., $p_{i_j} = similar(Q)$, then $selectivity(p_{i_j}) = selectivity_{shape\_similar}(Q)$. If p_{i_j} is a topological operator, i.e., $p_{i_j} = r(q_1, q_2, \theta)$, then $selectivity(p_{i_j}) = min\{selectivity_{shape\_similar}(Q_1), selectivity_{shape\_similar}(Q_2)\}$. If p_{i_j} is the complement of an operator, then its selectivity is estimated as the size of the DB minus the selectivity of the operator. To evaluate t_i, we first evaluate the p_{i_j} with the smallest selectivity. Then, we evaluate the rest of the operators on the images in the produced result. To compute the initial query, we just compute the union of the t_is.

6. GeoSIR: A Prototype System for Geometric-Similarity 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 [23] 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. A result of cluster detection is shown in Figure 11. There are seven clusters of shapes (A through G). The shapes are approximated by adequately small line segments, connected in polylines. Several heuristics may be used to minimize noise. Our similarity criterion 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.

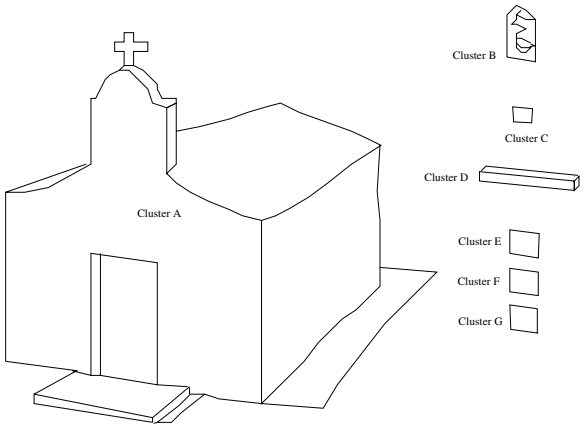

Figure 11. Seven clusters of shapes (A through G) have been detected.

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 de-

composition, 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. [22])) is an important direction for future research.

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.

7. 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. We are currently incorporating our method in a video retrieval system. Other future research directions include 3D awareness support.

References

[1] M. Ankerst, H. 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.

[2] L. Arge, V. Samoladas, and J. Vitter. On two-dimensional indexability and optimal range search indexing. In *PODS*, pages 346–357, Philadelphia PA, 1999. ACM.

[3] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):209–216, 1997.

[4] 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.

[5] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, and H.-P. Kriegel. Fast parallel similarity serach in multimedia databases. In *SIGMOD*, AZ, USA, 1997. ACM.

[6] A. D. 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.

[7] C. Bohm, B. Braunmuller, H.-P. Kriegel, and M. Schubert. Efficient similarity search in digital libraries. In *Advances in Digital Libraries*, Washington, DC, 2000. IEEE.

[8] G. Borgefors. An improved version of the chamfer matching algorithm. In *ICPR1984*, pages 1175–1177, 1984.

[9] G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, 1988.

[10] S. Cohen and L. Guibas. Shape-based image retrieval using geometric hashing. In *Proceedings of the ARPA Image Understanding Workshop*, pages 669–674, May 1997.

[11] E. R. Dougherty. *An Introduction to Morphological Image Processing*, volume TT9. SPIE press, 1992.

[12] R. Fagin and L. Stockmeyer. Relaxing the triangle inequality in pattern matching. *International Journal of Computer Vision*, to appear, 1999.

[13] 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.

[14] I. Fudos and L. Palios. An efficient shape-based approach to image retrieval. *Pattern Recognition Letters*, to appear, April 2002.

[15] J. E. Gary and R. Mehrotra. Similar shape retrieval using a structural feature index. *Information Systems*, 18(7):527–537, 1993.

[16] 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.

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

[18] 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.

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

[20] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proc. of the 22nd VLDB Coneference, Mumbai, India*, pages 215–226, 1996.

[21] R. Mehrotra and J. E. Gary. Similar-shape retrieval in shape data management. *IEEE Computer*, 28(9):57–62, 1995.

[22] M. Nabil, A. Ngu, and J. Shepherd. Picture similarity retrieval using the 2D projection interval representation. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):533–539, 1996.

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

[24] 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.

[25] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, to appear, November 2000.