The VLDB Journal 11: 17–27 (2002) / Digital Object Identifier (DOI) 10.1007/s007780100059

Efficient retrieval of similar shapes

Davood Rafiei[1], **Alberto O. Mendelzon**[2]

[1] Department of Computing Science, University of Alberta, Alberta, Canada; E-mail: drafiei@cs.ualberta.ca
[2] Department of Computer Science, University of Toronto, Toronto, Canada; E-mail: mendel@cs.toronto.edu

Edited by W. Klas. Received: 10 February 2000 / Accepted: 9 July 2001 /
Published online: 15 February 2002 – © Springer-Verlag 2002

Abstract. We propose an indexing technique for the fast retrieval of objects in 2D images based on similarity between their boundary shapes. Our technique is robust in the presence of noise and supports several important notions of similarity including optimal matches irrespective of variations in orientation and/or position. Our method can also handle size-invariant matches using a normalization technique, although optimality is not guaranteed here. We implemented our method and performed experiments on real (hand-written digits) data. Our experimental results showed the superiority of our method compared to search based on sequential scanning, which is the only obvious competitor. The performance gain of our method increases with any increase in the number or the size of shapes.

Keywords: Similarity queries – Image databases – Shape retrieval – Similarity retrieval – Fourier descriptors

1 Introduction

There is an increasing interest in storing and retrieving non-textual objects in databases. For example, this kind of data can be stored in the form of *extenders* in DB2, *DataBlades* in Informix, and *cartridges* in Oracle. Non-textual objects are frequently in the form of images or shapes. In cases where the key information for description or classification of an object can be found in its boundary, it is natural to store only the boundary and do retrieval based on that. Among the areas of applications for boundary shape matching are industrial inspection, object recognition in satellite images, character recognition, classification of chromosomes, and target recognition.

For example, consider the following query:

Query 1 Find all shapes similar to a given shape.

A basic question here is how we judge whether two shapes (for example the two shown in Fig. 1) are similar. There is a large body of work in the area of pattern recognition and computer vision on extracting boundary features of a shape and doing shape matching based on those features. The boundary of an object can be described in terms of simple descriptors such as length, diameter, and curvature ([MM86]), chain codes ([BG80,Bri81]), Fourier descriptors ([PF77,ZR72]) or moments ([BSA91]). Among these features, we use *Fourier descriptors* as our shape features. Theoretical and experimental evidence in favor of Fourier descriptors can be found in the literature [PF77,KSP95].

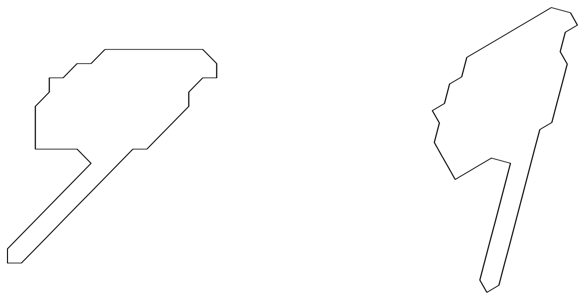
Fig. 1. Two shape boundaries both representing character ‘9’

Similar shapes often have differences in size and orientation. For example, consider the two shapes shown in Fig. 1. The *Euclidean distance* between their Fourier descriptors is 22.88. If we rotate the shape on the right by 30° in the clockwise (cw) direction, the Euclidean distance between their Fourier descriptors drops to zero. A simple approach to remove differences due to shifting, scaling, and rotation is to normalize Fourier descriptors before storing them in a database. However, there are still two problems with normalization. First, normalization is not guaranteed to minimize the distance between two arbitrary shapes. Second, normalization is not always desirable; for example, the shapes ‘9’ and ‘6’ should not be treated as similar if we are doing character recognition. A solution is to rewrite the query as follows:

Query 2 Find all shapes that become similar to a given shape after being rotated by $\theta \in [-30^\circ, 30^\circ]$.

If our shape collection includes, for example, shapes of airplanes, we may write our query instead as follows:

Query 3 Find all shapes similar to a given shape irrespective of rotation.

In this paper, we study the issue of efficiently processing these queries. We show how to organize Fourier descriptors in a multidimensional index, and how to efficiently use the index in processing a broad range of similarity queries. Our goal is to develop an access method that can handle shapes of various sizes and orientations, is much faster than sequential scanning, and does not miss any qualifying data objects in the answers (false positives are acceptable if they can be eliminated in a post-processing step without much performance degradation).

The organization of the rest of the paper is as follows. Section 2 provides some background material on related work, shape representation using Fourier descriptors and shape matching. In Sect. 3, we propose our technique for indexing shapes and processing similarity queries. Section 4 presents experimental results. We conclude in Sect. 5.

2 Background

2.1 Related work

The following relevant methods for multidimensional indexing and search have been proposed:

Jagadish [Jag91] proposes a technique for storing and retrieving shape descriptions in a multidimensional index. He maps shapes into their constituent rectangles, keeps a few larger rectangles in a multidimensional index, and uses the area difference between the constituent rectangles of shapes as a measure of similarity. Due to a normalization process, the shape description is invariant under translation and scaling. A problem with this approach is that a shape can be normally covered by multiple sets of rectangles. This can lead to ambiguity or storing multiple representations of the same shape. Furthermore, it is not possible to do matching in the presence of rotation; for example, two identical shapes may not match if one is rotated by $45°$.

Mehrotra and Gary [MG93] decompose a shape into several components and use fixed-sized segments of each component as the shape features. Based on a normalization process, the shape description is made invariant under translation, scaling, and rotation. A problem with this approach is that since a shape is broken down into pieces, the overall shape of the boundary is lost. In addition, each shape is described in terms of multiple feature vectors, and this introduces extra overhead during insertions and retrievals.

Berchtold et al. [BKK97] study the issue of storing polygons so that they can be retrieved based on partial similarity matches. They extract almost all possible boundary segments of polygons, transform each segment into a sequence of slope changes, and map the resulting sequences into their first few Fourier coefficients. Thus, each polygon is represented using a set of feature points, and the minimum bounding rectangle of these points for each polygon is stored in a multidimensional index. Due to a normalization, the shape representation is invariant to translation, scaling, and rotation, but it is not invariant to the starting point. This problem is handled by storing multiple descriptions of a polygon, each associated to a starting point. Again, representing a polygon in terms of multiple points introduces extra overhead during insertions and retrievals.

The QBIC (Query By Image Content) system [FBF$^+$94] contains a component for approximate shape matching. The system keeps a 20-D feature vector to describe the shape of every object identified in an image. Features, for example, include the area and the circularity, i.e., whether the object is circular or not. To allow fast retrieval, it is suggested to transform feature vectors using the Karhunen Loeve (KL) transform and keep a few important features (those associated with the few largest eigenvalues) in a multidimensional index. However, the choice of proper features and their weighting for each application is not an easy task. Some features are abstract quantities which may not easily fit in a distance function computation. In addition, the use of the KL transform makes the multidimensional index rather static.

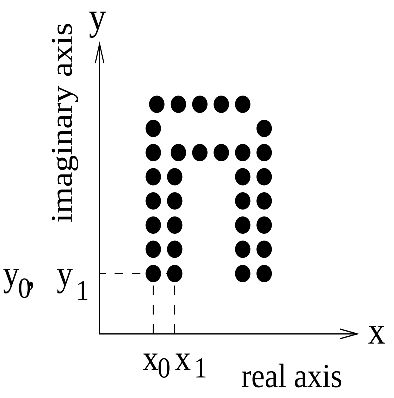

Fig. 2. A boundary and its representation as a complex sequence

The aforementioned methods are less general than ours because the notion of similarity is fixed before query evaluation; this notion cannot be changed unless a new index structure is created. Our method, instead, provides a set of transformations to express the notion of similarity in a query; yet, the resulting queries are evaluated using the same index, without prior knowledge of the specific transformations used. Therefore we have not compared the performance of our method with theirs, but with sequential scanning instead.

Related work on time series data includes the work of Agrawal et al. [AFS93] on using the discrete Fourier transform for retrieving similar time series and extensions and improvements over this approach [GK95,RM97,RM00]. Similar to our framework, Goldin and Kanellakis [GK95] show that the similarity retrieval will be roughly invariant to simple translations and scales if sequences are normalized before being stored in the index. The authors store in the index both the translation and the scale factors, in addition to normalized sequences, and also allow those factors to be queried using range predicates (see Goldin's Ph.D. thesis [Gol97] for implementation details).

A general framework for composing similarity queries is proposed by Jagadish, Mendelzon, and Milo [JMM95]. Our work here can be seen as a special case of this framework over shapes. Our shape matching can also be incorporated within a multimedia query language such as MOQL [LÖSO97] where multiple features of images are simultaneously queried.

2.2 Shape representation using Fourier descriptors

Given the figure of an object in the complex plane, its boundary can be traced producing a 1-D complex function b_t of time. For example, a point moving along the boundary shown in Fig. 2 generates the complex function $b_t = x_t + jy_t$ for

$t = 0, \ldots, N-1$ which is periodic with period N. That is, the x-axis of the figure is treated as the real axis and the y-axis as the imaginary axis of a sequence of complex numbers. Further information on tracing the boundary of a shape and possible alternatives in representing it can be found in any image processing textbook such as Gonzalez and Woods [GW92].

It should be noted that the description is solely based on the shape of the boundary; objects can still have holes in them, but this is not reflected in the description. Given a boundary function b_t, its Fourier transform can be written as

$$B_f = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} b_t e^{\frac{-j2\pi tf}{N}} \tag{1}$$

where $f \in \{\lfloor(N-1)/2\rfloor, \ldots, 0, \ldots, \lceil(N-1)/2\rceil\}$ and $j = \sqrt{-1}$ is the imaginary unit. The coefficients $B_0, B_{\pm 1}, \ldots$, called *Fourier descriptors*, describe the shape of the object in the frequency domain. The transformation is loss-less since the energy in the frequency domain is the same as the energy in the spatial domain (due to Parseval's theorem) and also the inverse Fourier transform gives the original boundary function.

2.3 Shape matching using Fourier descriptors

Consider two boundary functions $b_t = x_t + jy_t$ and $b'_t = x'_t + jy'_t$ (for $t = 0, \ldots, N-1$). A typical measure of similarity between the two boundaries is the *Euclidean distance*, which corresponds to mean-square error and which is also directly related to the cross-correlation [Raf98].

$$D^2(\boldsymbol{b}, \boldsymbol{b'}) = \sum_{t=0}^{N-1} |b_t - b'_t|^2 \tag{2}$$

However, the distance computation becomes ambiguous if the two boundaries have different numbers of samples. A solution to avoid this problem is to find the Fourier descriptors $\boldsymbol{B}$ and $\boldsymbol{B'}$, respectively, for $\boldsymbol{b}$ and $\boldsymbol{b'}$ and use a fixed number of lower frequency descriptors (say, $2M+1$) to compute the Euclidean distance, i.e.,

$$D^2(\boldsymbol{B}, \boldsymbol{B'}) = \sum_{f=-M}^{M} |B_f - B'_f|^2. \tag{3}$$

3 Our proposal

The general overview of the proposed method is as follows:

1. Obtain the Fourier descriptors of every shape boundary in the database.
2. Compute a fingerprint for every shape, as discussed in Sect. 3.1, and build a multidimensional index using the fingerprints. Each fingerprint is stored as a point in the multidimensional index.
3. For basic similarity queries (*proximity, nearest neighbours* and *all-pairs*), use the index to retrieve candidate shapes. The qualifying shapes are identified after retrieving their full database records and examining them.
4. For queries that use transformations in their expressions of similarities, if necessary, apply the transformations to the index, as discussed in Sect. 3.4, and retrieve candidate shapes. The full database record of every candidate shape is examined to find out if it qualifies.

We use Fourier descriptors as our shape features. Given a set of shape boundaries, for each boundary $\boldsymbol{b}$ we find its Fourier transform and retain only a fixed number of lower frequency descriptors. This number, which we denote by $2M+1$, can be chosen, for example to be the average length of a boundary in the spatial domain. If the number of Fourier descriptors happens to be less than $2M+1$, we store zero for higher frequency descriptors.

3.1 Computing a fingerprint

To aid in the retrievals that we intend to perform, we apply a few transformations to the descriptors, rather than storing them directly. First, we set B_0 to 0. B_0 is the only descriptor that carries information about the shape location. This setting minimizes the distance function (Eq. 3) with respect to translation. Next, the scale normalization is achieved by dividing every coefficient B_f by the amplitude of B_1, often called the fundamental frequency. $|B_1|$ turns out to be the largest amplitude when the boundary is traced in the counter-clockwise (ccw) direction and the boundary does not cross itself [WW80]. After the normalization, B_0 is 0, so we do not need to store it. Instead, we store the original value of B_0 before the normalization. It should be noted that the real and the imaginary parts of the initial value of B_0 represent the shift factors, respectively, along the X and the Y coordinates; the amplitude of the initial value of B_1 represents the scale factor. To totally get rid of B_1, which already has an amplitude of 1 for all shapes, we do an additional normalization. We shift the starting point such that the phase of B_1 becomes zero.

Definition 3.1. *Given the Fourier descriptors $B_{-M}, \ldots, B_M$ of a shape, denote the real part of B_0 by sh_x, the imaginary part of B_0 by sh_y, the amplitude of B_1 by sc, and the phase of B_1 by p. The shape description is defined as the sequence*

$$(sh_x, sh_y, sc, S_{-1}, S_2, S_{-2}, S_3, S_{-3}, \ldots, S_M, S_{-M}). \tag{4}$$

where $S_i = ((B_i - (sh_x + sh_y j))/sc) * e^{-ipj}$ (a complex number) for $i = -1, \pm 2, \pm 3, \ldots$.

The Euclidean distance between two shape descriptions, irrespective of variations in location and size, can be computed as follows:

$$D^2(\boldsymbol{S}, \boldsymbol{S'}) = \sum_{f=-M, f\neq 0,1}^{M} |S_f - S'_f|^2. \tag{5}$$

Such a description is still sensitive to changes in orientation and starting point of the tracing. We can assume that every data or query shape has a fixed starting point, if we encode its boundary using the same tracing algorithm and perform the same normalization. For example, a tracing algorithm may always start from the top right corner of a shape and trace it in the ccw direction. In this way, the starting point for two identical shapes will always be the same. Two similar shapes may still have small variations in their starting points, but those variations can be easily resolved by allowing some variations in starting points. This is discussed in Sect. 3.4.3.

There are sophisticated techniques to do phase normalization [PF77,WW80]. For example, Wallace et al. [WW80] suggest making the phases of the two coefficients of largest amplitude equal to zero. This is believed to shift the starting point over the axis of symmetry and also rotate the axis of symmetry such that it coincides with the real axis. However, it should be noted that none of these techniques are perfect in the sense that a shape can have two or more different phase normalizations, each as good as the others; or equivalently, two fairly similar shapes may have descriptors which are far from each other.

For the purpose of indexing, important features of the description need to be identified and placed in the fingerprint. First, changing the orientation or the starting point of a boundary only affects the phases of descriptors. To insulate the index from such changes, the information about the phases of descriptors is not stored in a fingerprint. Second, as is shown in Fig. 3, the lower frequency descriptors contain information about the general shape, and the higher frequency descriptors contain information about smaller details. There are strong reasons to believe that for a large class of boundary functions, the lower frequency descriptors contain most of the energy. For example, for continuous piece-wise smooth functions, the amplitude spectrum $|S_f|$ decreases at a rate proportional to f^{-2} [RH74, Page 373]. Thus, we can define a fingerprint of a shape as follows:

Definition 3.2. *Given a shape description* $(sh_x, sh_y, sc, S_{-1}, S_2, S_{-2}, \ldots, S_M, S_{-M})$*, the fingerprint of the shape is defined as* $(sh_x, sh_y, sc, |S_{-1}|, |S_2|, |S_{-2}|, \ldots, |S_k|, |S_{-k}|)$ *where* k *(*$\leq M$*) is the cut-off frequency.*

Next we show the completeness of the feature extraction.

3.2 Using fingerprints for indexing

The completeness of the indexing method is based on the following lemma:

Lemma 3.3. *The use of a fingerprint, in place of a full shape description for shape matching always returns a superset of the answer set.*

Proof: For every pair of boundaries $\boldsymbol{S}$ and $\boldsymbol{S'}$ of length $2M+1$ and for every $k \leq M$, we have

$$\sum_{f=-M, f\neq 0,1}^{M} |S_f - S'_f|^2 \geq \sum_{f=-k, f\neq 0,1}^{k} ||S_f| - |S'_f||^2 \qquad (6)$$

This is due to the fact that for every term $||S_f| - |S'_f||$ in the right side of the inequality, there is a term $|S_f - S'_f|$ in the left side and $|S_f - S'_f| \geq ||S_f| - |S'_f||$. $\square$

Thus, storing the fingerprints of shapes in the index does not affect the correctness since the index returns a superset of the answer set. Furthermore, the distance function on the right side of Eq. 6 is invariant to changes in the starting point of the boundary and rotation.

However, the index will not be effective if the choice of k results in a large number of false hits or high index dimensionality (*the curse of dimensionality*). Our experiments in Sect. 4.2 show that the value of k can be chosen as low as 2 which results in storing 5 Fourier amplitudes in the index.

There are a large number of multidimensional data structures which can be used for indexing (see the survey by Gaede and Günther [GG98] for details). We use the R*-tree as it is expected to work well for up to 20 dimensions and the length of a fingerprint is expected to be less than 20.

3.3 Basic similarity queries

Within this section, we assume that the shapes being compared have the correct sizes, positions, and orientations. Such a match can also be useful, for example before insertions, to prevent storing two replicas of the same image. We consider the three basic similarity queries over a shape database: (a) proximity query [1]; (b) all-pairs query; and (c) nearest-neighbours query.

In a proximity query, we are given a query shape and a threshold ϵ, and we would like to find all database shapes that are within distance ϵ of the query shape. To perform a proximity query, both the shape description and its fingerprint are computed as described in Sect. 3.1, in the same way as each data shape has been. The fingerprint is then used as a search key into the shape index, to retrieve all data shapes that are located in its proximity. Note that the index retrieves a superset of the answer set since it only keeps the fingerprints of shape descriptions. The actual result is obtained in an additional step where the Euclidean distance between the full database record of every matching shape and the query shape is computed.

In an all-pairs query, we are given two data sets and a threshold ϵ, and we want to find all pairs of shapes such that one shape is within distance ϵ of the other. To perform an all-pairs query, we do a spatial join between the corresponding indices of the two data sets. This is followed by an additional step where the Euclidean distance between the full database records of matching shapes are computed.

In a nearest-neighbours query, we are given a query shape, and we wish to find data shapes which are the closest to the query shape in distance. To perform a nearest-neighbours query, both the shape description and its fingerprint are computed (as discussed in Sect. 3.1), and the fingerprint is used as a search key over the index. Since the index employs the distance between fingerprints for its pruning and this distance is an underestimate of the real distance between descriptions, a nearest neighbour identified through searching the index may not be the real nearest neighbour. For example, of the two shapes *a* and *b*, *a* could be the closest to the query shape based on the distance between full descriptions, but the index will return *b* if *b* is the closest based on the distance between fingerprints.

To fix the problem, we pick the nearest neighbour(s) identified through the index and compute the distances between full descriptions of the retrieved shapes and the query shape. If we denote the minimum distance over all retrieved shapes with ϵ, the distance from the real nearest neighbours cannot

[1] This is often referred to as a *range query* as well [AFS93,LJF94].

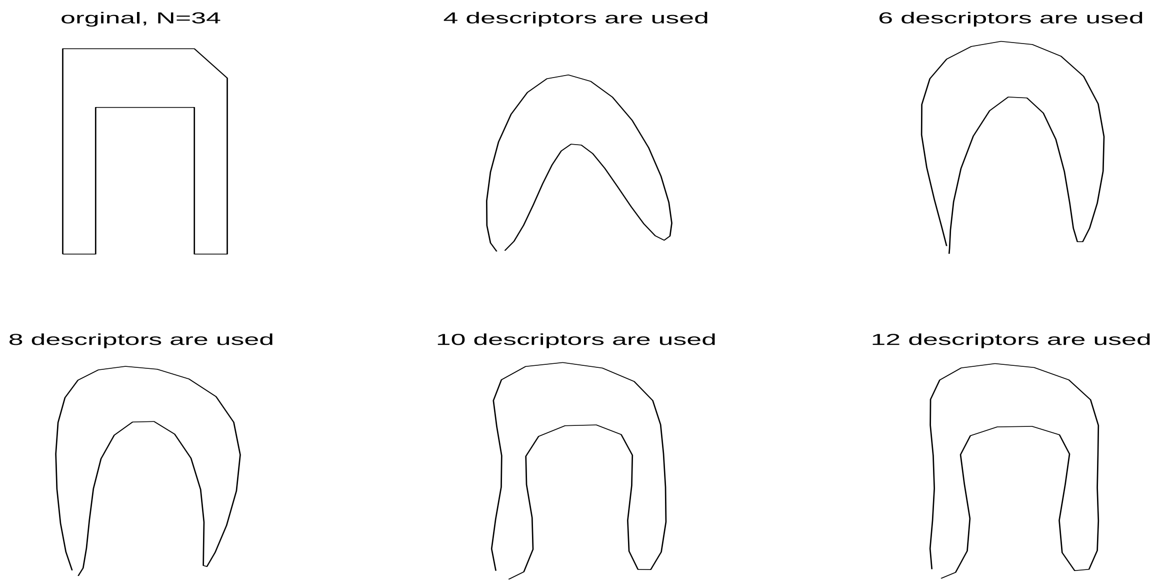

Fig. 3. Example of reconstructions from Fourier descriptors

be greater than ϵ; otherwise the shapes identified through the index are the nearest neighbours. The full algorithm is as follows:

Algorithm 1 :

1. Using a nearest-neighbours search algorithm (such as [RKV95]), retrieve the nearest neighbour(s) from the index.
2. For every candidate returned in step 1, retrieve its full database record and compute its real distance from the query shape. Let *NN* be the set of all data shapes at the minimum real distances from the query shape; let ϵ be this minimum distance.
3. Using ϵ as an initial threshold, pose an incremental proximity query to the index (results are returned one at a time and the threshold ϵ can be tightened during the process).
4. Get the next data shape within distance ϵ of the query shape. If the distance between the data shape and the query shape is less than ϵ, then set *NN* to be the new data shape and ϵ to be the new distance; if the distance between the new data shape and the query shape is ϵ, then add the new data shape to *NN*. Repeat this step until there are no more qualifying data shapes.

Algorithm 1 is a refinement of the nearest-neighbours algorithm given by Korn et al. [KSF+96]. The refinement is in the form of tightening the proximity query threshold in Step 4 as more data shapes are retrieved. There is another incremental refinement of the same algorithm, proposed by Seidl and Kriegel [SK98], which can also be used.

3.4 Queries with transformations

A natural way of doing shape matching is to remove certain differences before running a comparison. We can think of this process as applying certain transformations to images before doing a matching. We consider the following four kinds of transformations:

1. Shifting and scaling.
2. Rotation.
3. Change of starting point.
4. Smoothing.

In this section, we center our discussion on proximity queries, but the same techniques are applicable to nearest-neighbours and all-pairs queries.

Transformations 1 to 3 can be supported in a multidimensional index by providing a function that computes the distance between a data shape and a query shape; transformations can be applied to shape descriptions inside the function. Transformation 4 can be supported by registering an additional function that checks if an index entry overlaps with a query entry. The transformation can then be applied to either the index entry or the query entry (or both) before checking for an overlap. Most multidimensional index structures allow users to define such a function.

The next four subsections respectively discuss the evaluations of queries that use individual transformations 1 to 4 in their expressions of similarities. More details on evaluating queries that use a combination of transformations in their expressions of similarities can be found elsewhere [RM00].

3.4.1 Match with shifting or scaling

In many cases we do not care about the position of a shape within a coordinate system or about its size for matching purposes.

To match with shifting or scaling, a fingerprint is computed for the query shape, as described in Sect. 3.1, and this fingerprint is used as a search key for the index. If we are interested in a match invariant under shifting, we simply discard the shift factor of the query point and permit any value for the shift factor. Similarly, for scaling-invariant matching, we discard the scale factor of the query point and permit any value for the scale factor.

3.4.2 Match with rotation

We often wish to match shapes irrespective of small variations in orientation. For example, the two shapes shown in Fig. 1 make a perfect match, if one shape is rotated by $30°$. To achieve this, we state in our query the range of the rotation we wish to perform before doing a shape matching. Query 2, for instance, retrieves all database shapes that match a given query shape after one shape is being rotated by $\theta \in [-30°, 30°]$.

Sometimes, we would like to do matching totally invariant to rotation. For example, we may not care about the orientation at all if we are doing airplane recognition. This can be accomplished by simply allowing a rotation of $\theta \in [-180°, 180°]$ before matching.

To perform a match with rotation, a fingerprint is computed for the query shape and is used as a search key to the index. The search key is used to retrieve all candidates from the index. These candidates include all data points that match the query point irrespective of rotation factor. They also include false positives, i.e., data points that are not in the proximity of the query point for any rotation factor. To discard false positives, we need to retrieve the full database record of every candidate and check whether it actually falls in the proximity (say within distance ϵ) of the query shape after being rotated by some $\theta \in [\theta_1, \theta_2]$. On the other hand, rotating a shape boundary by θ is equivalent to multiplying every descriptor S_f by $e^{j\theta}$. We can thus rewrite Eq. 5 to make it reflect the rotation.

$$D^2(\boldsymbol{S}, \boldsymbol{S'}) = \sum_{f=-M, f\neq 0,1}^{M} |S_f - e^{j\theta}.S'_f|^2 \quad (7)$$

Lemma 3.4. *The minimum and the maximum of Eq. 7 take place at $\theta = \arctan(-X/Y) + c.\pi$ where c is an integer, $X = \sum \rho_f sin\psi_f, Y = \sum \rho_f cos\psi_f$ and $S^*_f.S'_f = \rho_f e^{j\psi_f}$ ($*$ denotes the complex conjugation [2]).*

Since we are interested in the minimum of Eq. 7 when $\theta \in [\theta_1, \theta_2]$ and $-\pi \leq \theta_1, \theta_2 \leq \pi$, the minimum must take place either at an endpoint (i.e., θ_1 or θ_2) or any point $\theta \in \{\arctan(-A/B) - \pi, \arctan(-A/B) + \pi, \arctan(-A/B)\}$ which is inside the region. It is straightforward to compute the distance function for these values and find out the optimal rotation factor that results in the minimum distance.

3.4.3 Match with changing starting point

When we compare two boundaries, we do not care about their starting points. If we use the same tracing algorithm for every boundary, there cannot be large variations in the starting point (though small variations are still possible). However, we may not have much control over the tracing algorithm, and as a result two similar shapes may have different starting points; or even if we use the same tracing algorithm for all boundaries, we may want to remove small variations (if any) before doing a comparison.

Shifting the starting point of a boundary by α [3] is equivalent to multiplying every descriptor S_f by $e^{jf\alpha}$. This operation, similar to rotation, only affects the phases of Fourier descriptors. Thus, we can still use the index to retrieve all candidates. To discard false positives, we need to retrieve the full database record of every candidate and check whether it still qualifies after the starting point is being shifted by some $\alpha \in [\alpha_1, \alpha_2]$. We can again rewrite Eq. 5 to make it reflect the shift in starting point.

$$D^2(\boldsymbol{S}, \boldsymbol{S'}) = \sum_{f=-M, f\neq 0,1}^{M} |S_f - e^{jf\alpha}.S'_f|^2 \quad (8)$$

The optimal value for α can be obtained by equating the derivative of the above equation to zero and finding the roots. This can be done using numerical techniques up to the machine precision [PTVF92].

3.4.4 Match with smoothing

Occasionally, we wish to do matching based on overall shape, irrespective of small variations in details and noise. In such cases, we would like to smooth out sharp edges and small variations before doing the comparison. To achieve this, we can apply a moving average transformation to shape boundaries. When an *l*-point moving average is applied to a boundary, every point is replaced with the average of its *l* surrounding points. On the other hand, applying a moving average to a boundary in the spatial domain corresponds to a vector multiplication in the frequency domain. For example, to apply a 2-point moving average to a boundary with 10 points, we can equivalently multiply its Fourier descriptors by the Fourier transform of vector $\boldsymbol{m_2} = (\frac{1}{2}, \frac{1}{2}, 0, 0, 0, 0, 0, 0, 0, 0)$. This gives us the Fourier descriptors of the smoothed boundary.

A distinguishing feature of smoothing, compared to other transformations discussed in this paper, is that its effect on a shape depends on the characteristics of the shape. This is unlike rotation, for instance, where the effect of rotating a data shape by θ before a comparison is the same as that of rotating the query shape by $-\theta$.

Given a query shape and a desired moving average for smoothing, the matching can be performed as follows:

1. Find the Fourier transform of the desired moving average (as demonstrated for 2-point moving average); let us denote this by $\boldsymbol{M}$.
2. *Transforming the query shape*: Apply the transformation to the query shape description $(sh_x, sh_y, sc, \boldsymbol{Q})$ by replacing $\boldsymbol{Q}$ with $\boldsymbol{Q'}$ where $Q'_i = Q_i * M_i$ for $i = -1, -2, 2, \ldots, -k, k$.
3. Construct a search key by computing the fingerprint of the new shape description.
4. *Transforming the index*: Apply $\boldsymbol{M}$ to data entries stored in the index before checking for an overlap between a data entry and the search key; this is done inside the function that checks if a data entry from the index overlaps the search key.
5. For every candidate, retrieve its full database record, apply $\boldsymbol{M}$ to it and check if the resulting shape falls in the proximity of $\boldsymbol{Q'}$.

The transformation can be applied to the index on the fly as the index is being traversed. The issue of on-the-fly applying single or multiple transformations to an index is studied in the domain of time series data [RM97,RM00]. The same techniques can be applied to the domain of shapes.

[2] The complex conjugate of $z = x + yj$ is defined as $z^* = x - yj$.

[3] For example, $\alpha = 2\pi s_0/N$ for a boundary of length N means shifting its starting point by s_0 points in ccw direction.

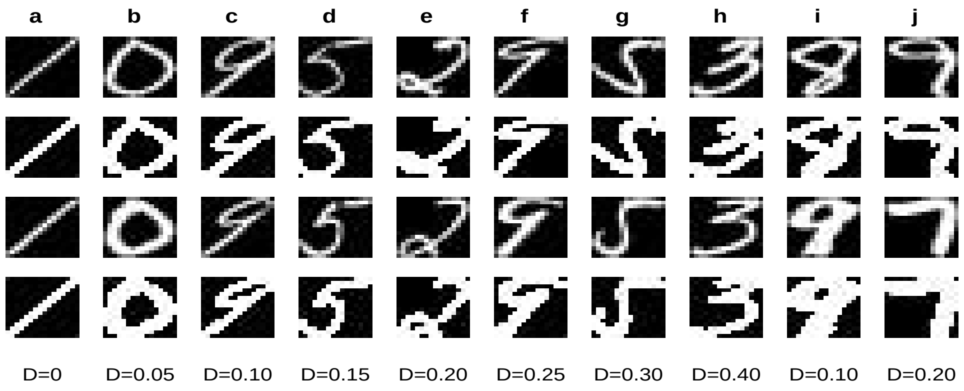

Fig. 4. Query shapes, shown in the top two rows, and their nearest neighbours, shown in the bottom two rows

4 Experiments

To determine the effectiveness of our proposed technique, we implemented our method and ran experiments on a dataset of 11,000 real hand-written digits. The data was obtained from the CEDAR CDROM dataset, which was gathered from scanned ZIP codes at the Buffalo Post Office [Hul94]. For every digit, the dataset held 1,100 images. Each image was originally in the form of a 16×16 gray-scale image which was converted into a binary image (by thresholding) and was traced to identify a shape boundary. Then, the boundary was encoded using 30 lower Fourier descriptors. For boundaries with length less than 30, zero was padded at the end. For each shape, both its description and its fingerprint are computed, as outlined in Sect. 3.1, and used for the purpose of indexing. As our index, we used Norbert Beckmann's implementation of the R*-tree [BKSS90]. For the nearest-neighbours search, we implemented the algorithm developed by Roussopoulos et al. [RKV95] as part of Algorithm 1 over R*-tree. We stored 10,000 shapes (1,000 samples of each digit) in the index and used the 1,000 remaining samples as queries. We ran each query 10 times and averaged the execution times from these runs. All our experiments were conducted on a 168 MHz Ultrasparc station.

We investigated the following questions:

- How effective and practical is our technique in classifying shapes in a real data domain?
- How many Fourier coefficients should we store in the index? Storing larger number of coefficients reduces the number of false positives but increases the index dimensionality, and as a result the search time.
- How does our technique compare to sequential scanning?

4.1 Shape classification

To verify the effectiveness of our proposed technique in classifying shapes, we tried to classify all 1,000 query shapes by assigning every query shape to the class of its nearest neighbours. When there was more than one nearest neighbours for a shape, we picked one randomly. The result was interesting: 96.4% of shapes were classified correctly. Some of those query shapes are shown, in their gray scale and binary representation, in the two top rows of Fig. 4 along with their nearest neighbours shown in the two bottom rows of the same figure.

Table 1. Various ranges of rotations and their effects in correctly classifying the shapes of hand-written digits

Rotation factor $\theta \in$	Fraction of query shapes classified correctly (%)
$[0, 0]$	96.4%
$[-10, 10]$	96.5%
$[-20, 20]$	96.4%
$[-30, 30]$	96.4%
$[-40, 40]$	96.3%
$[-50, 50]$	96.3%

As is shown, query shapes shown in Figs. 4a to 4h are classified correctly with their Euclidean distances from their nearest neighbours varying from 0 to 0.40. The query shape shown in Fig. 4i is not classified correctly, but its binary representation looks quite similar to that of its nearest neighbour. The query shape shown in Fig. 4j looks different from its nearest neighbour, though their boundaries still look similar.

In another experiment, we used Query 2 and tried to identify for each query shape its nearest neighbour irrespective of a rotation factor $\theta \in [-30°, 30°]$. This did not change the overall classification rate, i.e., only 96.4% of shapes were classified correctly. However, allowing a rotation factor in general did retrieve better matches. Figure 5 shows six query shapes (in the top two rows), their original nearest neighbours (in the middle two rows) and their optimal nearest neighbours (in the bottom two rows) when the rotation factor varied from $-30°$ to $30°$. As is shown, for example rotating the data shape shown at the bottom of Fig. 5a by $11°$ in the ccw direction reduces its Euclidean distance from the query shape to 0.30; this is less than the Euclidean distance between the same query shape and its original nearest neighbour. Table 1 summarizes the effect of various rotations in correctly classifying shapes. As is shown, applying a small rotation ($\theta \in [-10, 10]$) to data shapes before matches slightly improves the classification rate of hand-written digits; larger rotations, on the other hand, either have no effect or deteriorate the rate of correct classifications. This is because the digit data is generally sensitive to orientations and allowing larger rotations can potentially retrieve more non-identical digits.

We later picked 1,000 shapes among those stored in the database, applied to each shape a random rotation in the range $[-\pi, \pi]$ and used it as a query shape. We only specified the

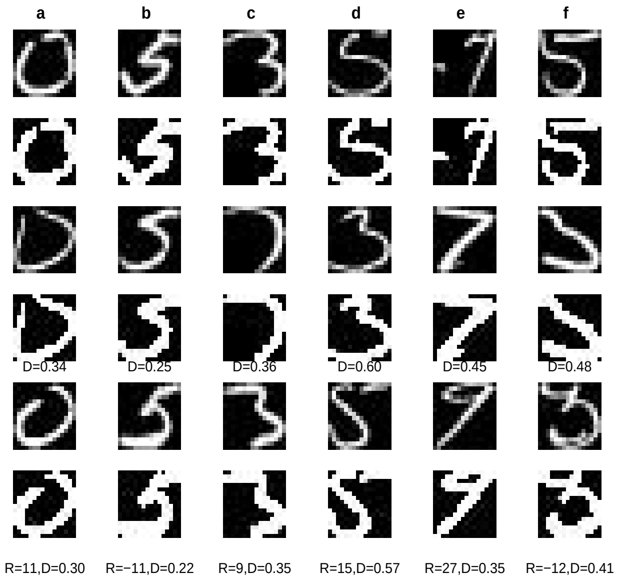

Fig. 5. Query shapes (the top two rows), their original nearest neighbours (the middle two rows) and their optimal nearest neighbours (the bottom two rows) varying the rotation factor in $[-30°, 30°]$

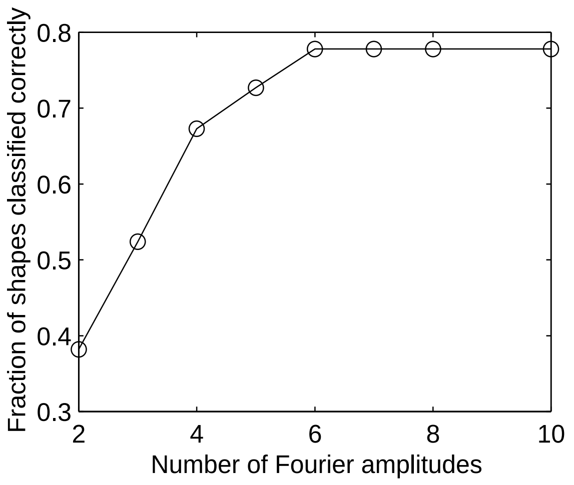

Fig. 6. The fraction of query shapes classified correctly, varying the number of Fourier amplitudes used for classification

rotation interval in our query. As expected, for each query shape, only the shape itself was retrieved from the database.

4.2 Varying the cut-off frequency

The effectiveness of the index mainly depends on the concentration of the key shape information within a few descriptors of fingerprints. To measure this effectiveness, we ran some experiments varying the number of Fourier descriptors stored in a fingerprint. Figure 6 shows the ratio of query shapes that are classified correctly (according to the criteria outlined in Sect. 4.1) to all query shapes varying the number of Fourier amplitudes used for classification. As the number of amplitudes increases up to 6, the ratio of shapes that are classified correctly increases accordingly up to 0.778. This ratio remains the same despite increasing the number of Fourier amplitudes from 6 to 10. Compared to a full shape description which consists of both the amplitudes and the phases of 30 lower Fourier coefficients, classifying 96.4% of the shapes correctly, a fingerprint does a pretty good job using only 6 amplitudes which make up only 10% of a full shape description and still classifying 0.778% of the shapes correctly.

Figure 7a shows the average execution time of Algorithm 1 for 1,000 nearest-neighbours queries, broken into: (1) search time in Step 1 to identify the initial approximate nearest neighbours; and (2) search time in Step 3 to find the real nearest neighbours. Figure 7b shows the fraction of index nodes accessed, averaged over 1,000 nearest-neighbours queries, again broken into the fractions accessed in Step 1 and Step 3.

As the number of Fourier amplitudes increases, the index selectivity improves, i.e., the index gives fewer false hits. The number of false hits, as is depicted in Fig. 8 for a proximity query, mainly depends on the number of Fourier amplitudes used in fingerprints and the output size of the query. Due to the high similarity between different shapes of the same digit, a large fraction of our false hits (for example, 62% when the output size was 11 and the number of Fourier amplitudes was 6) were other shapes of the same digit depicted by the query shape which were not within the specified distance of the query shape.

The reduction in false hits reduces the search time since less time is needed to remove those false hits. However, increasing the number of Fourier amplitudes after some point, often called the *cut-off frequency*, either does not reduce the number of false hits or reduces it only slightly. This is because higher frequency amplitudes carry less of the energy than lower frequency ones. On the other hand, the search time increases with the index dimensionality, because the tree becomes deeper. Furthermore, the pruning becomes harder, as is shown in Fig. 7 with the ratio of index nodes that are accessed, because the probability of an arbitrary data bounding rectangle being close to the query point increases with the dimensionality.

Given the trade-off between the tree search time and the time spent for removing false hits, it is natural to expect that there is an optimal cut-off frequency. Based on our experiments, as illustrated in Figs. 6 and 7, the optimal cut-off frequency occurs for as few as 6 Fourier amplitudes.

4.3 Comparison to sequential scanning

Figure 9 shows the average execution time of our proposed method compared to sequential scanning for 1,000 nearest-neighbours queries. To get its best performance, we used buffered input for sequential scanning, in a system with buffer size of 8,192 bytes. For the experiment shown in Fig. 9a, the border length was fixed to 30 while the database size varied from 10,000 to 30,000 shapes. Since the size of dataset was limited, we doubled or tripled the size by adding one or two randomly rotated copies of each shape to the database. This doubling did not affect the performance of sequential scanning, which was linear in the input size, but we expected the doubling to deteriorate the performance of our method since high similarity among database shapes would increase the number of false hits. For the experiment shown in Fig. 9b, the number of shapes was fixed at 10,000 while the number of Fourier descriptors used to represent a boundary varied from

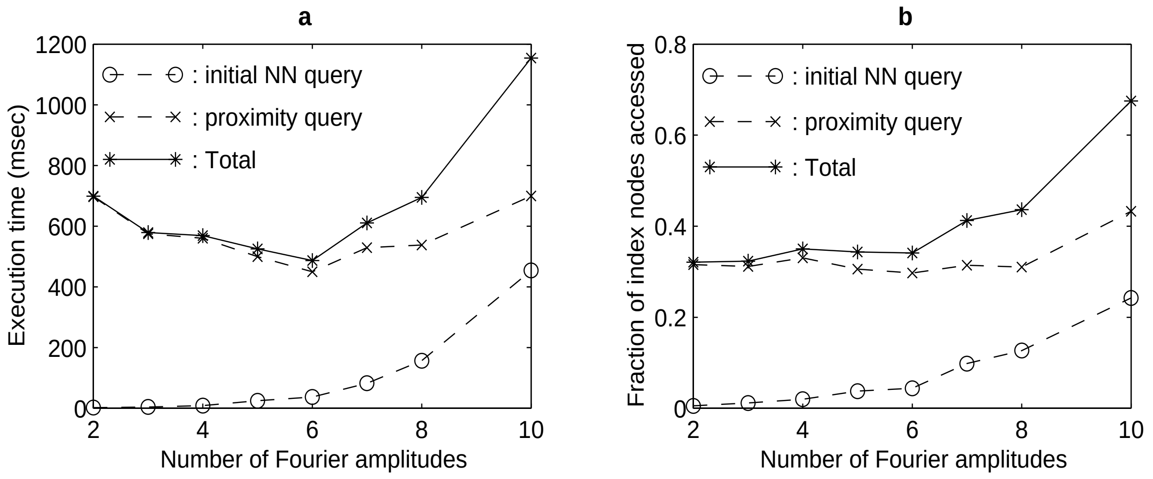

Fig. 7. Break up of **a** the execution time and **b** the fraction of index nodes accessed, for nearest-neighbours queries, varying the number of Fourier amplitudes

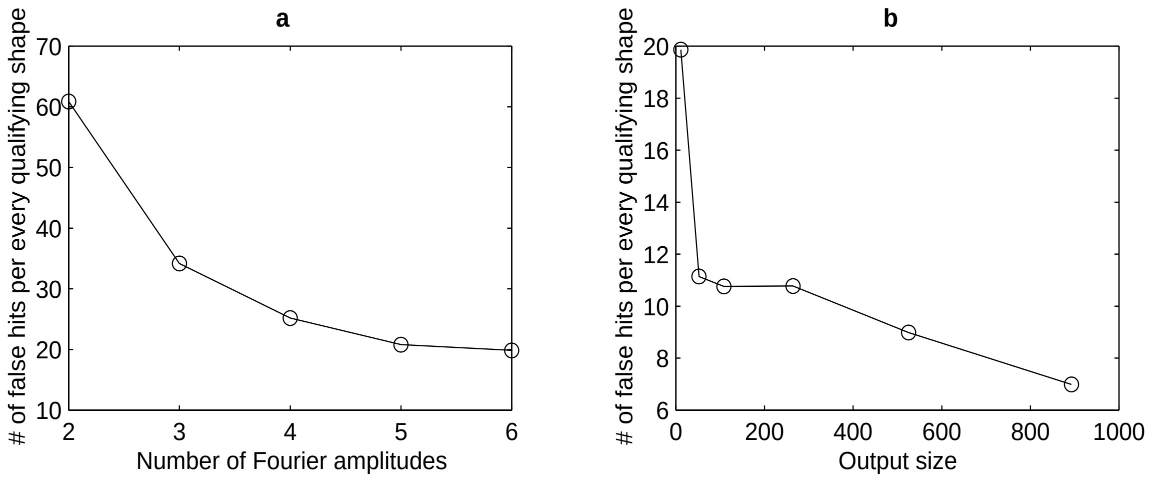

Fig. 8. The average number of false positives for every qualifying shape **a** varying the number of Fourier amplitudes and fixing the average output size of the query to 11, and **b** varying the average output size and fixing the number of Fourier amplitudes to 6

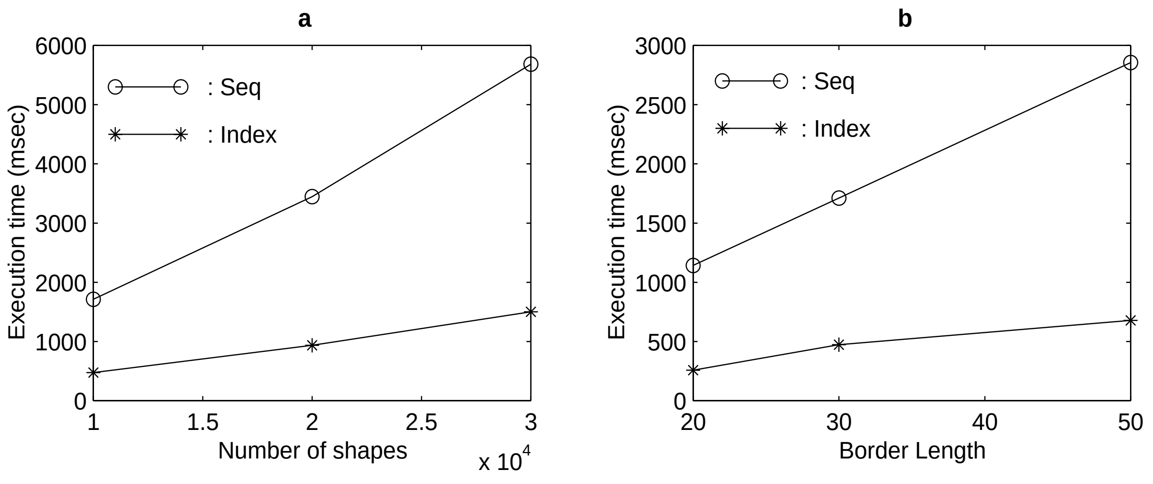

Fig. 9. **a** Time per query varying the number of shapes, for nearest-neighbours queries. **b** Time per query varying the border length, for nearest-neighbours queries

20 to 50. As shown in the figure, increasing either the number of shapes or the border length increases the relative advantage of our method, making it more attractive for large databases.

5 Conclusions

We have proposed an indexing technique that can efficiently retrieve images of objects based on similarity between their boundary shapes. We have used Fourier descriptors as our shape features and have developed an index organization such that similar shapes can be easily retrieved irrespective of their differences in size, position and orientation. The highlight of our contribution is an index structure that helps find optimal matches between shapes irrespective of various differences between them. Our technique has the following desirable properties:

- It uses a shape matching mechanism which is well-studied in the area of pattern recognition.
- It exploits the fact that important features of a large class of shapes are concentrated within only a few Fourier descriptors.
- It can handle shapes of various sizes.
- It guarantees efficient retrieval of all qualifying shapes.

Furthermore, we have presented a refinement of an earlier nearest-neighbours search algorithm for feature vectors that are truncated, due to the significance of some features over others, before being stored in a R-tree index.

Acknowledgement. We thank the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems for their support of this work.

References

[AFS93] Agrawal R, Faloutsos C, Swami A (1993) Efficient similarity search in sequence databases. In: Proc. 4th International Conference on Foundations of Data Organizations and Algorithms (FODO '93), pp 69–84, Chicago

[BG80] Bribiesca E, Guzman A (1980) How to describe pure form and how to measure differences in shape using shape numbers. Pattern Recognition 12(2):101–112

[BKK97] Berchtold S, Keim DA, Kriegel HP (1997) Using extended feature objects for partial similarity retrieval. VLDB J 6(4):333–348

[BKSS90] Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R* tree: an efficient and robust index method for points and rectangles. In: Proc. ACM SIGMOD International Conference on Management of Data, pp 322–331, Atlantic City

[Bri81] Bribiesca E (1981) Arithmetic operations among shapes using shape numbers. Pattern Recognition 13(2):123–138

[BSA91] Belkasim SO, Shridhar M, Ahmadi M (1991) Pattern recognition with invariants: a comprehensive study and new results. Pattern Recognition 24:1117–1138

[FBF$^+$94] Faloutsos C, Barber R, Flickner M, Niblack W, Petkovic D, Equitz W (1994) Efficient and effective querying by image content. J Intell Inf Syst 3(3/4):231–262

[GG98] Gaede V, Günther O (1998) Multidimensional access methods. ACM Comput Surv 30(2):170–231

[GK95] Goldin DQ, Kanellakis PC (1995) On similarity queries for time-series data: constraint specification and implementation. In: 1st Int. Conference on the Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 976. Springer, Berlin Heidelberg New York, pp. 137–153

[Gol97] Goldin DQ (1997) Constraint query algebras. PhD thesis, Brown University, www.cs.brown.edu/people/dgk/Papers/thesis.ps

[GW92] Gonzalez RC, Woods RE (1992) Digital image processing. Addison-Wesley, Reading, Mass., USA

[Hul94] Hull JJ (1994) A database for handwritten text recognition research. IEEE Trans Pattern Anal Mache Intell 16(5):550–554

[Jag91] Jagadish HV (1991) A retrieval technique for similar shapes. In: Proc. ACM SIGMOD International Conference on Management of Data, pp 208–217, Denver, Colo., USA

[JMM95] Jagadish HV, Mendelzon AO, Milo T (1995) Similarity-based queries. In: Proc. 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp 36–45, San Jose, Calif., USA

[KSF$^+$96] Korn F, Sidiropoulos N, Faloutsos C, Siegel E, Protopapas Z (1996) Fast nearest neighbor search in medical image databases. In: Proc. 22nd International Conference on Very Large Data Bases, pp 215–226, Mumbai, India

[KSP95] Kauppinen H, Seppanen T, Pietikainen M (1995) An experimental comparison of autoregressive and Fourier-based descriptors in 2D shape classification. IEEE Trans Pattern Anal Mach Intell 17(2):201–207

[LJF94] Lin KI, Jagadish HV, Faloutsos C (1994) The TV-tree - an index structure for high-dimensional data. VLDB J 3(4):517–542

[LÖSO97] Li JZ, Özsu MT, Szafron D, Oria V (1997) MOQL: a multimedia object query language. In: Proc. 3rd International Workshop on Multimedia Information Systems, pp 19–28

[MG93] Mehrotra R, Gary JE (1993) Feature-based retrieval of similar shapes. In: Proc. 9th International Conference on Data Engineering, pp 108–115, Vienna, Austria

[MM86] Mokhtarian F, Mackworth A (1986) A scale-based description and recognition of planar curves and two dimensional shapes. IEEE Trans Pattern Anal Mach Intell 8(1):34–43

[PF77] Persoon E, Fu KS (1977) Shape discrimination using Fourier descriptors. IEEE Trans Syst Man Cybern 7(2):170–179

[PTVF92] Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1992) Numerical recipes in C. Cambridge University, Cambridge, UK

[Raf98] Rafiei D (1998) Fourier-transform based techniques in efficient retrieval of similar time sequences. PhD thesis, University of Toronto

[RH74] Richard CW, Hemami H (1974) Identification of three-dimensional objects using Fourier descriptors of the boundary curve. IEEE Trans Syst Man Cybern 4:371–378

[RKV95] Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proc. ACM SIGMOD International Conference on Management of Data, pp 71–79, San Jose, Calif., USA

[RM97] Rafiei D, Mendelzon AO (1997) Similarity-based queries for time series data. In: Proc. ACM SIGMOD International Conference on Management of Data, pp 13–24, Tucson, Ariz., USA

[RM00] Rafiei D, Mendelzon AO (2000) Querying time series data based on similarity. IEEE Trans Knowl Data Eng 12(5):675–693

[SK98] Seidl T, Kriegel HP (1998) Optimal multi-step nearest neighbour search. In: Proc. ACM SIGMOD International Conference on Management of Data, pp 154–165, Seattle, Wash., USA

[WW80] Wallace TP, Wintz PA (1980) An efficient three-dimensional aircraft recognition algorithm using normalized fourier descriptors. Comput Graph Image Process 13:99–126

[ZR72] Zahn CT, Roskies RZ (1972) Fourier descriptors for plane closed curves. IEEE Trans Comput 21(3):269–281