Efficient Image Retrieval through Vantage Objects*

Jules Vleugels Remco Veltkamp

Department of Computer Science, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

Abstract

We describe a new indexing structure for general image retrieval that relies solely on a distance function giving the similarity between two images. For each image object in the database, its distance to a set of m predetermined *vantage objects* is calculated; the m-vector of these distances specifies a point in the m-dimensional *vantage space*. The database objects that are similar (in terms of the distance function) to a given query object can be determined by means of an efficient nearest-neighbor search on these points. We demonstrate the viability of our approach through experimental results obtained with a database of about 48,000 hieroglyphic polylines.

1 Introduction

Recent years have seen a growing interest in developing effective methods for searching large image databases. While manual browsing may be adequate for collections of a few hundred images, larger databases require automated tools for search and perusal. Content-based image retrieval [13, 19–21] is based on certain characteristics of the images; our particular interest lies in approaches based on feature extraction [18]. Such methods perform matching on the contents of the image itself, by comparing features of the query image with those of images stored in the database. *Global features* are derived from the entire shape; examples of this are roundness, central moment, eccentricity, and major axis orientation [2, 7]. We are specifically interested in features with a geometric flavor, such as being able to search a collection of vector images for the occurrence of a query polyline.

In this paper, we present a way of efficiently comparing the features of a given query image to a large number of images stored in a database. Our approach works by mapping database objects onto points in an m-dimensional space, in such a way that points that lie close together correspond to images with similar features. This allows us to efficiently retrieve objects that are similar to a given query object by determining the points that lie close to the point corresponding to the query image. The only requisite for our approach is that a similarity distance function be defined on the database objects.

As an application, we implemented our approach on a database of about 48,000 hieroglyphic polylines [1], and show how to efficiently retrieve the hieroglyphics with polylines that are similar (under translation and rotation) to those of a given query hieroglyphic.

1.1 Related work

Wolfson [25] describes the Geometric Hashing paradigm, which enables a unified approach to rigid object recognition under different viewing transformations for both two- and three-dimensional objects. It is assumed that we can extract the interest points of a given model. In an intensive

*This research was supported by SION project No. 612-21-201: Advanced Multimedia Indexing and Searching (AMIS).

preprocessing stage, this model information is indexed into a hash table using transformation-invariant features. Given a query image, its interest points are extracted and efficiently compared to those stored in the database. A good (partial) match votes for the possible presence of a stored model; by not requiring a perfect match, some of the interest points can be absent or hidden by other objects without affecting the matching algorithm. The candidate models thus obtained are then checked in a computationally-expensive verification step. A drawback of this paradigm lies in its time and storage bounds. A query with an objects that contains m so-called *interest points*—for example, for a polyline, m would typically be the number of vertices—has a worst-case complexity of $O(m^3)$, and for n planar objects with m interest points each the hash table requires $O(nm^3)$ storage to achieve invariance under translation, rotation, and scaling.

The vantage-object structure we describe in this paper is a paradigm to store objects in a data structure such that objects with similar features can be retrieved in an efficient manner. Wolfson [25] describes a different paradigm for rigid-object recognition under different viewing transformations, which he calls *Geometric Hashing*. All combinations of so-called *interest points* of the database objects are indexed into a hash table. The interest points of a given query image are compared to those stored in the database through a voting mechanism. A drawback of this paradigm lies in its time and storage bounds: a query with an object containing m interest points—for example, the number of vertices for polylines—has a worst-case complexity of $O(m^3)$, and for n planar objects with m interest points each the hash table requires $O(nm^3)$ storage to achieve invariance under translation, rotation, and scaling.

An important ingredient to the approach described in this paper is an algorithm that determines nearest neighbors among a set of high-dimensional points. Several theoretically efficient solutions [4, 5, 9, 14] have arisen from the Computational Geometry community, but these algorithms are not always equally efficient in practice. Yianilos [26] considers the problem of finding nearest neighbors in general metric spaces. He introduces the vantage-point tree (*vp-tree* for short) together with associated algorithms. (Uhlmann [23] has independently reported the same structure, calling it a *metric tree*.) The expected complexity of a vp-tree query is under certain circumstances $O(\log n)$, and in practice it appears to perform somewhat better than a k-d tree. The vp-tree is however not guaranteed to be balanced, so its expected complexity may not be met for inputs with particular unfavorable distributions. It should be emphasized that, despite the similarity in naming, our vantage-object structure has only little in common with the vantage-point tree: our vantage-object structure is a data structure used to store objects for efficient similarity retrieval, whereas the vp-tree implements nearest-neighbor queries on point sets. The similarity in naming occurs because both structures categorize database objects in terms of their distances from vantage objects.

For completeness, we mention some other nearest-neighbor data structures with similar properties that have been devised with high-dimensional data sets in mind: the SR-tree due to Katayama and Satoh [12], the SS-tree of White and Jain [24], the TV-tree proposed by Lin et al. [15], and the X-tree as reported by Berchtold et al. [6].

The nearest-neighbor step of our algorithm can be replaced by a range search, that is, instead of determining the k nearest neighbors, we could consider all points that lie within some fixed distance from the query point. (This is essentially what is achieved by the distance threshold we introduce in Section 4.4.) There exists a wealth of literature on range searching [8, 16, 17]; recent research has considered the problem of achieving good asymptotic complexity as well as favorable running times. The algorithm due to Schwarzkopf and Vleugels [22], which—besides being theoretically efficient—has been explicitly designed to perform well in practice for realistic inputs.

1.2 Paper outline

We first describe some notations and conventions used throughout this paper. In Section 3 we describe the framework of our vantage-object structure; this framework leaves some choices to be filled in for specific applications. Section 4 describes the application of our framework to the test case of retrieving images from a database of hieroglyphics, for which experimental results are

provided in Section 5. Finally, we conclude the paper and indicate directions of future work in Section 6.

2 Preliminaries

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of n objects in $\mathbb{R}^2$. In this paper, we call a continuous function $d : \mathcal{A} \times \mathcal{A} \mapsto \mathbb{R}$ a *distance function* on $\mathcal{A}$ if and only if for all $A_i, A_j, A_k \in \mathcal{A}$ with $1 \leqslant i, j, k \leqslant n$:

1. $d(A_i, A_i) = 0$ (one-way identity),
2. $d(A_i, A_j) \geqslant 0$ (positive), and
3. $d(A_i, A_j) \leqslant d(A_i, A_k) + d(A_k, A_j)$ (triangle inequality).

Note that we neither require a distance function to have the usual identity nor to be symmetric, that is, $d(A_i, A_j) = 0$ does not necessarily imply that $A_i = A_j$, and moreover $d(A_i, A_j)$ need not be equal to $d(A_j, A_i)$. However, the distance function defined by d should closely adhere to our intuitive notion of shape resemblance for the results of a query to be perceived as resembling the query object.

Throughout the paper, a superscripted asterisk * is used for variables that are related to vantage objects; a subscripted question mark $_?$ applies to query-related variables.

3 The vantage object structure

Suppose we are given a collection of objects with a distance measure d defined on them. Consider two objects A_1 and A_2 that are very similar—that is, $d(A_1, A_2)$ is small—, and let A^* be some third object, which we call a *vantage object*. Since d satisfies the triangle inequality, $|d(A_1, A^*) - d(A_2, A^*)|$ will be small as well. In other words, we can measure the resemblance between A_1 and A_2 by comparing their respective distances from A^*. Note that this correspondence is strictly one way: objects that are similar will have similar distances from a vantage object A^*, but objects with similar distances are not necessarily similar in appearance.[1]

Given a query object $A_?$, we can thus determine (a superset of) all similar objects by computing the distance of $A_?$ to some vantage object A^* and selecting all objects that achieve similar distance from A^*. More formally, each object A_i corresponds to a point p_i in the one-dimensional *vantage space* defined by A^*, in which the coordinate of p_i is given by $d(A_i, A^*)$. As noted above, this selection will also include objects that are not similar to $A_?$ but happen to have similar distance to A^*. Moreover, if the set of objects is large, the set of objects with similar distance will most likely grow large as well. Both of these problems can be relieved by increasing the number of vantage objects to, say, m. Let $\mathcal{A}^* = \{A_1^*, \ldots A_m^*\}$ be a set of m vantage objects. Each object $A_i \in \mathcal{A}$ corresponds to a point $p_i = (x_1, \ldots, x_m)$ in the m-dimensional vantage space, such that $x_j = d(A_i, A_j^*)$. For a given query object $A_?$, we compute its distance to each of the m vantage objects; this defines a point $p_?$. Next, we determine all vantage-space points that are sufficiently close to $p_?$; each of these points corresponds to an object of $\mathcal{A}$.

Once again, it is not guaranteed that each of the objects returned is similar to the query object; however, all objects that are sufficiently similar to $A_?$ are correctly returned in this manner. To see this, define the *object-space neighbors* $\mathrm{nbors}(A_?, \epsilon)$ of $A_?$ as the set of objects whose distance from $A_?$ is at most ϵ:

$$\mathrm{nbors}(A_?, \epsilon) = \{A_i \in \mathcal{A} \mid d(A_i, A_?) \leqslant \epsilon\}.$$

In contrast, let $\mathrm{nbors}^*(A_?, \mathcal{A}^*, \epsilon)$ denote the set of *vantage-space neighbors* of $A_?$, that is, the objects whose distance from each of the vantage objects differs by at most ϵ from the corresponding distance of $A_?$:

$$\mathrm{nbors}^*(A_?, \mathcal{A}^*, \epsilon) = \{A_i \in \mathcal{A} \mid \forall A^* \in \mathcal{A}^* : |d(A_i, A^*) - d(A_?, A^*)| \leqslant \epsilon\}.$$

[1] This is not difficult to see by considering the equivalence to the natural numbers: even though the distance from 1 to 7 equals that from 13 to 7, the distance from 1 to 13 is much larger.

Lemma 3.1 *The set of vantage-space neighbors of a query object $A_?$ includes its object-space neighbors, that is,* $\text{nbors}^*(A_?, \mathcal{A}^*, \epsilon) \supseteq \text{nbors}(A_?, \epsilon)$.

Proof. Let A_i be an object with $d(A_i, A_?) \leqslant \epsilon$, and $\mathcal{A}^* = \{A_1^*, \ldots, A_m^*\}$ the set of vantage objects. By the triangle inequality, $d(A_i, A_j^*) \leqslant d(A_i, A_?) + d(A_?, A_j^*) \leqslant d(A_?, A_j^*) + \epsilon$ holds for each j with $1 \leqslant j \leqslant m$. It follows that A_i is included in the set $\text{nbors}^*(A_?, \mathcal{A}^*, \epsilon)$ as well. □

A similarity query among objects thus reduces to a simple nearest-neighbors query among a set of points. To answer such a query, we need only compute the distance of the query object to the m vantage objects, and determine the vantage-space points that are sufficiently close to the resulting query point. The main advantage of this approach is that the computationally most-expensive step—computing the similarity measure between the database objects—is performed entirely offline; at runtime we can deal with points rather than the original images.

This framework leaves some details to be filled in. For one, it is defined in terms of some distance function and a set of vantage objects, both of which depend on the application at hand. Secondly, we need a means of finding the k nearest neighbors of a query point among a set of m-dimensional points. For now, we limit this discussion to mentioning that several known solutions [4, 5, 9, 14] achieve $O(k \log n)$ query time after $O(n \log n)$ preprocessing. This gives the following result.

Theorem 3.2 *Let $\mathcal{A}$ be a set of n objects, $A_?$ a query object, and $\epsilon \geqslant 0$ a constant, and let T denote the time required to compute the distance between any two given objects. For any constant $m > 0$, we can preprocess $\mathcal{A}$ in $O(mnT + n \log n)$ time and $O(mn)$ space, such that we can retrieve the objects A_i with $d(A_i, A_?) \leqslant \epsilon$ in $O(mT + k \log n)$ time, where k is the cardinality of* $\text{nbors}^*(A_?, \mathcal{A}^*, \epsilon)$.

4 Matching hieroglyphics

We implemented the algorithm described in the previous section on a collection of approximately 4,700 hieroglyphics with a total of over 48,000 polylines [1]. To apply the framework to this database, we need a matching distance function that measures the distance between two hieroglyphics, and pick a number of vantage objects for the collection. Additionally, the framework requires an efficient algorithm to retrieve the nearest vantage-space neighbor of a given hieroglyphic. Finally, each hieroglyphic consists of a number of polylines, and we need a way to map the query results for the separate polylines to entire hieroglyphics.

4.1 A matching distance function.

Arkin et al. [3] describe a matching metric for polygons that is invariant under translation, rotation, and scaling, is defined for both convex and non-convex polygons, and can be computed in time $O(c_1 c_2 \log c_1 c_2)$, where c_1, c_2 are the numbers of vertices of the respective polygons. The metric is based on the L_2 distance between the turning functions of two polygons. The *turning function* $\Theta_A(s)$ of a polygon measures the angle of the counterclockwise tangent as a function of the arc length s, measured from some reference point O on the boundary of A. In other words, $\Theta_A(0)$ is the angle v that the tangent at the reference point O makes with some reference orientation, for example, the x-axis; $\Theta_A(s)$ keeps track of the turning that takes place as one traces the boundary of A, increasing with left-hand turns and decreasing with right-hand turns.

Turning functions are periodic: $\Theta_A(s) = \Theta_A(s + l) \bmod 2\pi$, where l is the perimeter length of A. The turning function $\Theta_A(s)$ of a polygon A is translation invariant by definition, and becomes $\Theta_A(s) + \alpha$ if the polygon is rotated over α degrees; scaling invariance can be achieved by normalizing the polygons to a standard perimeter length of 1. The distance between two turning

functions Θ_A and Θ_B is defined as

$$\min_{0 \leqslant t < 1,\ 0 \leqslant \alpha < 2\pi} \int_0^1 |\Theta_A(s) - \Theta_B(s+t) + \alpha|\ ds.$$

In our approach we essentially use the same metric, with some slight adaptations to the case of matching polylines rather than polygons. For one, note that the distance of a sufficiently short line segment l from any polyline A is zero since l will perfectly match any (longer) line segment of A. We overcome this problem by enforcing a threshold on the number of segments of the database polylines; only polylines with a sufficient number of segments are considered. Also, we cannot simply scale the polylines to some unit length because we want to be able to match portions of a polyline rather than matching a polyline in its entirety. For example, consider a polyline P and a longer polyline Q that contains P as a sub-polyline. Since P occurs in its entirety in Q, we want the latter to perfectly match the former. If we would scale both polylines to some unit length, the (turning functions of the) polygons would no longer match very well. In other words, we cannot simply scale the two polylines prior to matching to achieve scale invariance. At the moment we do not have a viable alternative to this, which implies that our implementation is not scale invariant. This could be overcome by either adapting the above metric to scale invariance or employing a different distance function, such as the one proposed by Cohen and Guibas [10].

Our matching algorithm runs in identical $T = O(c_1 c_2 \log c_1 c_2)$ time for two polylines, with c_1, c_2 as before. This brings the entire preprocessing step for n polylines with m vantage polylines to $O(mnc^2 \log c)$, where c is the maximum number of segments to a single polyline; this is $O(mn)$ if we consider c a constant. (In the hieroglyphics database, c is about 400. While this may seem a relatively high number, the average is much lower: about fourteen.)

4.2 Choosing vantage objects.

Another aspect of our approach is the choice of vantage objects. Ideally, the m vantage objects should differentiate the database objects as well as possible. This means that they should measure different 'properties' of the objects; if the vantage objects are not very different from one another, the distance from an object to each vantage object is similar, and little information is gained by adding the extra vantage objects. A possible way to ensure that each vantage point adds relevant discriminating power is by choosing the vantage points such that they lie maximally far apart. Computing the k furthest among a set of points, however, is an exponential problem; we resort to the following heuristic algorithm instead. First, choose an initial vantage object—polyline, in our case—A_1^* at random. Next, repeatedly pick the next vantage object A_i^* by maximizing the minimum distance to the previously chosen vantage objects $A_1^*, \ldots, A_{i-1}^*$. While the vantage objects thus chosen will probably not be maximally far apart, they will at least constitute a set that is spread out well.

As for the number of vantage polylines we should choose, there is a clear trade-off: fewer polylines will return more non-relevant polylines as answers to a query, whereas more polylines increase the dimension of the search space and, therefore, the time required to answer a query. Some comparisons for both different values of m and different vantage objects are given in Section 5.

4.3 Retrieving nearest neighbors.

The problem of determining nearest neighbors among a set of points is well studied in the field of Computational Geometry, and several efficient solutions exist [5, 9, 14]. Of particular interest is the approximate-nearest-neighbor algorithm due to Arya et al. [4], who show that the nearest-neighbor problem can be solved particularly efficiently if we weaken the problem formulation to determining approximate nearest neighbors to the query point. For any query point $q \in \mathbb{R}^d$ and constant $\epsilon > 0$, a point $p \in P$ is a $(1+\epsilon)$-*approximate nearest neighbor* of q if, for all $p' \in P$, we have $d(p,q) \leqslant (1+\epsilon) \cdot d(p',q)$. A set of n points in $\mathbb{R}^d$ can be preprocessed in $O(n \log n)$ time and $O(n)$ space, such that given a query point $q \in \mathbb{R}^d$, a constant $\epsilon > 0$, and a constant integer $k \geqslant 1$, we can compute $(1+\epsilon)$-approximations to the k nearest neighbors of q in $O(k \log n)$ time. Besides

being theoretically optimal in the worst case, their approach has been observed to be very efficient in practice, even for $\epsilon = 0$.

For m vantage polylines, a single query among n polylines thus takes $O(mT + k \log n)$ time, where k is the number of polylines returned, and T is again the time required to compare any two given polylines. With the distance function described previously, this is $O(mc^2 \log c + k \log n) = O(m + k \log n)$, where c is the (constant) maximum number of segments to a polyline.

As a postprocessing step, we explicitly measure the similarity between the query polyline and each of the k polylines returned by the algorithm; this takes additional $O(kc^2 \log c) = O(k)$ time, and thus does not influence the given bound. This postprocessing step has been omitted in the experimental results presented here since our main goal it to demonstrate the usefulness of the vantage-object structure by itself, even without further postprocessing of the query results.

4.4 Matching entire hieroglyphics

So far, our algorithm description focused on matching single polylines. The hieroglyphics in our database however consist of several—typically about ten—separate polylines. Therefore, we need to devise a strategy to combine the results for multiple polylines into a single result for each hieroglyphic.

Given a query hieroglyphic consisting of h polylines, we perform h separate queries, one for each polyline. The polylines returned by these queries are grouped according to the hieroglyphic they are part of; this gives a certain number of matching polylines for each hieroglyphic. Next, the hieroglyphics should be presented to the user in a way that reflects their resemblance to the query hieroglyphic. The most effective way would be to explicitly measure their distance from the query polygon. As mentioned before, for the purpose of this paper we refrain from doing so, because we want to demonstrate that our vantage-object retrieval algorithm already provides a relevant ordering of the query results. Instead, we sort the hieroglyphics matched using the following simple heuristics. The more polylines of a given hieroglyphic match closely with a polyline of the query hieroglyphic, the better we consider the hieroglyphic to match the query. We sort hieroglyphics with an identical number of matching polylines according to increasing maximum distance of a matched polyline from the corresponding query polyline.

Note that it is not a priori clear how many nearest neighbors we want to retrieve for each query point, since the grouping by hieroglyphic is not done until after the retrieval step. We chose to use a threshold on the maximum distance of each vantage-space point from the query point; the polylines for which this distance does not exceed the threshold are returned as answers to the queries.

5 Experimental results

We implemented the algorithm described in the previous sections, using an available implementation of the search algorithm due to Arya et al. [4], on a 400 MHz Pentium II-based workstation.

Some typical queries for $m = 6$ vantage points, along with the hieroglyphics returned, are shown in Table 1. For these queries, we set the minimum number of segments that a polyline should consist of to three—only polylines that define at most a single angle are ignored in the matching process. The three queries increase in detail. The first one consists of a simple shape that recurs in numerous hieroglyphics; all query results contain a—more or less, as the hieroglyphics have been digitized by hand—exact copy of the shape. The second query is more specific, and as a result we get some matches that contain an exact copy of the query bird, followed by a number of hieroglyphics that contain similarly-shaped birds. The last query shown contains a shape that is unique within in the database, and the query results resemble the query object to various degree. (Note that the intuitive resemblance seems to gradually decrease with the ranking.)

As mentioned, the bulk of the computations are performed during the preprocessing steps. For each polyline in the database we need to compute the corresponding vantage-space point, which involves computing its distance from each of the vantage polylines; this takes about twenty-three

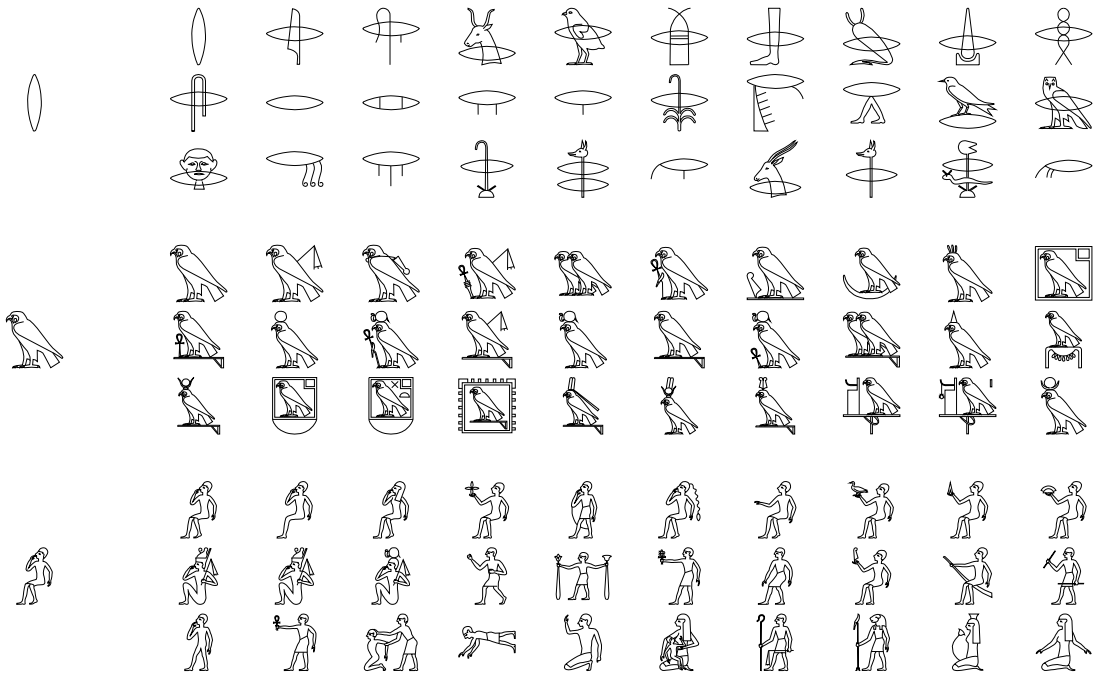

Table 1: Three example queries together with their 30 best matches.

minutes per vantage polyline. In case we want to add polylines to the database, this computation need only be performed for the newly added polylines. The time required for building the nearest-neighbor data structure depends on the number m of vantage objects, but is, for example, only slightly more than three seconds for $m = 10$. This step has to be performed only when the image database changes. Finally, the query time is low due to the intensive preprocessing steps; for example, for 1000 randomly selected query polylines and $m = 6$ vantage objects, retrieving the 100 nearest neighbors takes 5.4 ms on average.

For comparison purposes, we also implemented a trivial algorithm using the same implementation of the matching algorithm. To find the database polylines similar to a given query polyline, we compute the distance of each polyline in the collection from the query and take the best-matching one. Answering a query for a single polyline this way is identical to computing the distance from a vantage polyline, and therefore also takes somewhat over twenty minutes. To match an entire query hieroglyphic, this time should be multiplied by the number of polylines the hieroglyphic consists of.

Another issue is the number of vantage objects. Table 2 summarizes the number k of results returned, as well as the computing time t^* required for a single query, for different values of m and ϵ, where m is the number of vantage points and ϵ is the distance threshold. The overall tendency is that the query time does not increase significantly—that is, more than approximately linear—with an increasing number of vantage objects. for example, the average time of 5.41 ms for $m = 6$ mentioned above becomes 1.78 ms for $m = 2$. The number of results returned, however, quickly grows if we do with less vantage points, which means that more (possibly expensive) postprocessing will be required to present the results in some meaningful order. Finally, another benefit of using more vantage points is that the relevance of the query results appears to be much better if we increase the number of vantage points.

	$\epsilon = 0.05$	$\epsilon = 0.10$	$\epsilon = 0.25$	$\epsilon = 0.50$	$\epsilon = 1.00$	
m	k	k	k	k	k	t^*
1	3312	6387	15010	24944	> 30000	0.40 ms
2	76.8	244	1261	3833	11692	1.78 ms
3	8.8	54.2	375	1182	4742	2.79 ms
4	3.0	18.4	195	817	3021	3.04 ms
6	1.2	8.2	71.8	483	1911	5.41 ms
8	1.0	3.2	46.8	215	1338	6.13 ms
10	1.0	1.2	34.8	164	1073	8.82 ms

Table 2: Comparison for different numbers of vantage points and distance thresholds.

6 Conclusions

We presented an indexing structure for general image retrieval that relies solely on a distance function giving the similarity between two images, and demonstrated that it can be used to efficiently determine shape-based image similarity.

At the moment, our approach is invariant only under translation and rotation; we are still working on scale invariance. A possible workaround is to perform queries with a number of differently scaled versions of the query object, but a scale-invariant distance function would obviously be preferable.

Although the viability of our approach was demonstrated only for a single specific case—retrieving hieroglyphic images—its potential is far more general. In fact, our indexing structure applies to any set of images for which a similarity distance function can be defined, which includes raster images as well as vector images. Since almost all distance calculation are performed during an offline preprocessing step, an advantage of our approach is that it does not rely on the distance function being cheap and/or easy to compute; a query requires only a small number—typically ten or less—of distance calculations. Therefore, if sufficient preprocessing time is available, one can use an elaborate (and possibly expensive) distance function with desirable properties such as robustness against noise, blurrings, cracks, and deformations [11].

References

[1] The extended library. Centre for Computer-Aided Egyptological Research, Faculty of Theology, Utrecht University, Utrecht, the Netherlands. `http://www.ccer.theo.uu.nl/ccer/extlib.html`.

[2] Edoardo Ardizzone, Marco La Cascia, Viti Di Gesú, and Cesare Valentie. Content based indexing of image and video databases by global and shape features. In *Proc. Int. Conf. Pattern Recognition*, 1996.

[3] Esther M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and Joseph S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):209–216, 1991.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994. An implementation is available from `http://www.cs.umd.edu/~mount/ANN`.

[5] J. L. Bentley. K-d trees for semidynamic point sets. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 187–197, 1990.

[6] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for higher dimensional data. In *Proc. 22th VLDB Conference*, pages 28–39, 1996.

[7] M. La Cascia and E. Ardizzone. JACOB: Just a content-based query system for video databases. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 1996.

[8] Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[9] K. L. Clarkson. Nearest neighbor queries in metric spaces. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 609–617, 1997.

[10] S. D. Cohen and Leonidas J. Guibas. Partial matching of planar polylines under similarity transformations. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 777–786, January 1997.

[11] M. Hagedoorn and R. C. Veltkamp. Measuring resemblance of complex patterns. In *Proc. Int. Conf. Discrete Geom. Comput. Imagery*, 1999.

[12] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD '97*, pages 369–380, 1997.

[13] P. M. Kelly, T. M. Cannon, and D. R. Hush. Query by image example: the CANDID approach. In *Proc. SPIE: Storage and Retrieval for Image and Video Databases III*, volume 2420, pages 238–248, 1995.

[14] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimension. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 599–608, 1997.

[15] K. I. Lin, H. V. Jagdish, and C. Faloutsos. The TV-tree: An index structure for higher dimensional data. *VLDB Journal*, 4:517–542, 1994.

[16] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[17] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993.

[18] Rajiv Mehrotra and James E. Gary. Similar-shape retrieval in shape data management. *IEEE Computer*, 28:57–62, 1995.

[19] W. Niblack, R. Barber, W. Equitz, M. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content using color, texture and shape. *Storage Retrieval Image Video Databases*, 1908:173–187, 1993.

[20] Virginia E. Ogle and Michael Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28:40–48, 1995.

[21] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. In *Proc. SPIE: Storage and Retrieval for Image and Video Databases II*, volume 2185, pages 34–47, 1994.

[22] Otfried Schwarzkopf and Jules Vleugels. Range searching in low-density environments. *Inform. Process. Lett.*, 60:121–127, 1996.

[23] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inform. Process. Lett.*, 40:175–179, 1991.

[24] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. 12th IEEE Internat. Conf. Data Engineering*, pages 516–523, 1996.

[25] H. J. Wolfson. Model-based object recognition by geometric hashing. In *Proc. 1st Europ. Conf. Comp. Vision*, pages 526–536, 1990.

[26] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 311–321, 1993.