PERGAMON

Pattern Recognition 35 (2002) 69–80

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Efficient image retrieval through vantage objects[☆]

Jules Vleugels*, Remco C. Veltkamp

Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, 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 two image databases, one consisting of about 5200 raster images of stamps, the other containing about 72,000 hieroglyphic polylines. © 2001 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Image retrieval; Indexing; Data structure; Matching; Vantage objects

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 [1–4] is based on certain characteristics of the images; our particular interest lies in approaches based on feature extraction [5]. 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.

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 two databases. The first consists of about 5200 color JPEG images of Dutch stamps; we demonstrate that our algorithm is capable of efficiently retrieving similarly colored images. The second database contains about 72,000 hieroglyphic polylines, and we 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

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. Lamdan et al. [6] and Wolfson [7] describe a different paradigm for rigid-object recognition under different viewing transformations, which they call *geometric hashing*. All combinations of the so-called *interest points* of the database objects are indexed into a hash table. The interest points of a given query image

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

*Corresponding author. Tel.: + 31-30-2534110; fax: + 31-30-2513791.

E-mail addresses: jules@cs.uu.nl (J. Vleugels), remco.veltkamp@cs.uu.nl (R.C. Veltkamp).

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 [8–11] have arisen from the computational geometry community, but these algorithms are not always equally efficient in practice.

An early approach is due to Burkhard and Keller [12]. They pose the problem of searching a file containing a set of keys for the one closest to a given query key, and present three file structures together with their corresponding search algorithms. Though the term *vantage* is not yet coined in their paper, it can be considered one of the first vantage-based approaches since the set of keys is clustered into a number of sets according to their distances from an arbitrary (but fixed) element.

Yianilos [13] 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 [14] 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. An important difference is that in the vp-tree a distance calculation is performed for every node visited during a search, of which there are $O(\log n)$ to $O(n)$ in the worst case. In our approach we first perform a small constant number of distance calculations, after which only $O(\log n)$ simple coordinate comparisons are required for retrieving the nearest neighbors. This difference becomes especially important if the distance function is expensive to compute.

For completeness, we mention some of the other nearest-neighbor data structures with comparable properties that have been devised with high-dimensional data sets in mind. The X-tree was introduced by Berchtold et al. [15] to improve on some shortcomings of the R-tree, especially for high-dimensional data. The SS-tree of White and Jain [16] uses bounding spheres for the shape of minimum bounding regions (MBR) and allows for exact k-nearest-neighbor queries on high-dimensional data sets, usually outperforming other structures such as the R*-tree and the kd-B-tree. While based on the same idea, the SR-tree (sphere/rectangle tree) introduced by Katayama and Satoh [17] gains an increase in performance by representing MBRs as the intersection of a bounding sphere and a bounding rectangle. In contrast, the idea of the TV-tree proposed by Lin et al. [18] is to refrain from fully defining each MBR. Instead it uses a so-called telescopic minimum bounding region (TMBR) that is specified only in a few dimensions — the so-called active dimensions — and extends infinitely in all other dimensions. This allows for more efficient storage, since only a few features (for example, coordinates) of the data needs to be stored. Additional features are used only when their additional discriminatory power is absolutely necessary.

Alternatively, the nearest-neighbor step of our algorithm can be replaced by a range search: 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 5.3.) There exists a wealth of literature on range searching [19–21]; recent research has considered the problem of achieving good asymptotic complexity as well as favorable running times. The algorithm due to Schwarzkopf and Vleugels [22,23] has been explicitly designed to — besides being theoretically efficient — perform well in practice for realistic inputs.

1.2. Paper outline

We first describe the notations and conventions used throughout this paper. Then 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 test case of retrieving similarly colored images from a database of stamp raster images using this framework. Another application, namely that of retrieving images from a collection of hieroglyphics, is described in Section 5. Finally, we conclude the paper and indicate directions of future work in Section 6.

2. Preliminaries

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

(i) $d(A_i, A_i) = 0$ (one-way identity),

(ii) $d(A_i, A_j) \geqslant 0$ (positiveness), and
(iii) $d(A_i, A_k) \leqslant d(A_i, A_j) + d(A_j, A_k)$ (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 $d(A_j, A_i)$. (When comparing two polylines, for example, one polyline may be contained in the other — thus giving a small value of d — but not vice versa.) We refer to $d(A_i, A_j)$ as *the distance of* A_j *from* A_i. The distance function defined by d should closely adhere to our intuitive notion of 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.

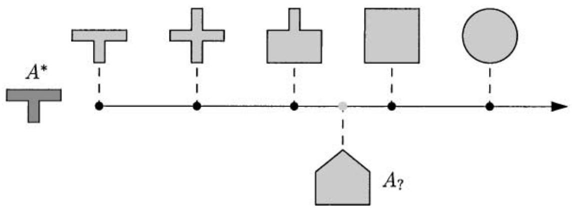

Fig. 1. A vantage object imposes an ordering on the database objects.

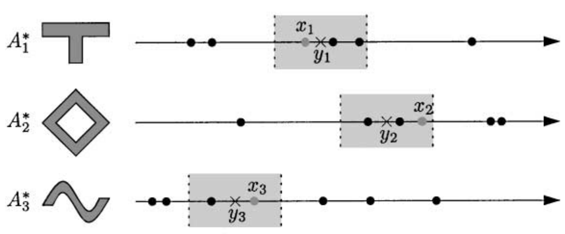

Fig. 2. Multiple vantage objects generate multiple orderings.

3. The vantage object structure

Suppose we are given a collection of objects with a distance measure d defined on them. The idea is to compute the similarity of each object from some fixed object, which we call a *vantage object*. This naturally imposes on the objects an ordering on decreasing similarity to the vantage objects, as illustrated in Fig. 1. The objects shown at the top are ordered by decreasing similarity to the vantage object A^* shown at the left. The query object $A_?$ (shown at the bottom) can now be placed within this ordering by likewise computing its similarity to the vantage object. Objects with similar features end up at similar positions within the ordering, thus reducing similarity matching to a simple nearest-neighbor search.

More formally, consider an object A_1 that closely matches another object A_2 — that is, $d(A_1, A_2)$ is small — and let A^* be some vantage object. Since d satisfies the triangle inequality $d(A^*, A_2) \leqslant d(A^*, A_1) + d(A_1, A_2)$, we know that $d(A^*, A_2) - d(A^*, A_1)$ is small as well. In other words, we can measure the resemblance between A_1 and A_2 by comparing their 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 $A_?$ from the 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 $d(A^*, A_i)$; we return the set $\{A_i\}$ of objects for which $d(A^*, A_i)$ is sufficiently small as possible matches. Two problems may however occur. For one, as noted above the selection will 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.

3.1. Multiple vantage objects

The two problems addressed above can be relieved by increasing the number of vantage objects. Let $\mathscr{A}^* = \{A_1^*, \ldots, A_m^*\}$ be a set of m vantage objects. As before, each vantage object imposes a one-dimensional ordering on the database objects, as illustrated in Fig. 2. For each object A_i, the list of A_j^* contains exactly one corresponding point p_j. Likewise, a query object $A_?$ defines m points q_j, one for each vantage object. Any object resembling the query object achieves similar positions in each of the lists because its distance from each vantage objects will be similar to that of the query object. Thus we may consider the object A_i to resemble the query object only if p_i occurs close to q_i in each of the m lists. Note that this not only lessens the likelihood of returning an object that does not resemble the query (but by coincidence achieves similar distances from the vantage object); it reduces the number of objects returned as possible matches as well.

[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.

In formal terms, each object $A_i \in \mathscr{A}$ corresponds to a point $p_i = (x_1, \ldots, x_m)$ in the m-dimensional vantage space with $x_j = d(A_j^*, A_i)$. For a given query object $A_?$ we compute its distance from each of the m vantage objects; this likewise defines a point $p_? = (y_1, \ldots, y_m)$. Next, we determine all vantage-space points that are sufficiently close to $p_?$; each of these points corresponds to an object of $\mathscr{A}$.

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 prove this claim, we consider two objects A_i and A_j to be similar if $d(A_i, A_j) \leqslant \varepsilon$ and $d(A_j, A_i) \leqslant \varepsilon$ (for some given value of ε), and define the *object-space neighbors* $\text{nbors}(A_?, \varepsilon)$ of $A_?$ as the set of objects A_i similar to a query object $A_?$:

$$\text{nbors}(A_?, \varepsilon) = \{A_i \in \mathscr{A} \mid d(A_?, A_i) \leqslant \varepsilon \wedge d(A_i, A_?) \leqslant \varepsilon\}.$$

In contrast, let $\text{nbors}^*(A_?, \mathscr{A}^*, \varepsilon)$ 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 $A_?$:

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

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

Proof. Let $\mathscr{A}^* = \{A_1^*, \ldots, A_m^*\}$ be the set of vantage objects, and A_i an object that is included in $\text{nbors}(A_?, \varepsilon)$; we have $|d(A_?, A_i)| \leqslant \varepsilon$. For each $A^* \in \mathscr{A}^*$ the triangle inequality gives

$$d(A^*, A_i) \leqslant d(A^*, A_?) + d(A_?, A_i) \leqslant d(A^*, A_?) + \varepsilon$$
$$\Leftrightarrow d(A^*, A_i) - d(A^*, A_?) \leqslant \varepsilon$$

and similarly

$$d(A^*, A_?) \leqslant d(A^*, A_i) + d(A_i, A_?) \leqslant d(A^*, A_i) + \varepsilon$$
$$\Leftrightarrow d(A^*, A_?) - d(A^*, A_i) \leqslant \varepsilon.$$

Combining these inequalities gives $|d(A^*, A_i) - d(A^*, A_?)| \leqslant \varepsilon$, proving that A_i is included in the set $\text{nbors}^*(A_?, \mathscr{A}^*, \varepsilon)$ as well. □

A similar query among objects thus reduces to a simple range-searching query among a set of points (see Fig. 3). 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. The algorithm is summarized below.

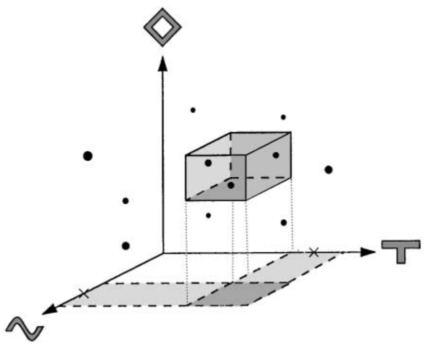
Fig. 3. A query among multiple vantage objects corresponds to range searching in a higher-dimensional space.

Algorithm *Retrieval using Vantage Objects*

```
/* preprocessing */
1:  forall $A_i \in \mathscr{A}$ do
2:      forall $A_j^* \in \mathscr{A}^*$ do $x_j \Leftarrow d(A_j^*, A_i)$ end for
3:      $p_i \Leftarrow (x_1, \ldots, x_m)$
4:  end for

/*  query */
5:  while (true)
6:      $A_? \Leftarrow$ a query object
7:      for all $A_j^* \in \mathscr{A}^*$ do $y_j \Leftarrow d(A_j^*, A_?)$ end for
8:      $q \Leftarrow (y_1, \ldots, y_m)$
9:      return $\{A_i \in \mathscr{A} \mid |p_i - q| \leqslant \varepsilon\}$
10: end while
```

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 determining the points that lie within a distance of ε from a given point. For now, we limit this discussion to mentioning that several known solutions [8–11] achieve $O(k \log n)$ query time after $O(n \log n)$ preprocessing, where k is the number of points retrieved. This gives the following result.

Theorem 3.2. *Let $\mathscr{A}$ be a set of n objects, $\mathscr{A}_?$ a query object, and $\varepsilon \geqslant 0$ 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 $\mathscr{A}$ in $O(mnT + n \log n)$ time and $O(mn)$ space, such that we can retrieve the objects A_i with $d(A_?, A_i) \leqslant \varepsilon$ in $O(mT + k \log n)$ time, where k is the cardinality of* $\text{nbors}^*(A_?, \mathscr{A}^*, \varepsilon)$.

4. Histogram matching: stamps

We implemented the algorithm described in the previous section and applied it to the problem of retrieving images based on color information. For this we used a collection of 5246 stamps from the Netherlands and the Dutch colonies, consisting of color JPEG images of typically about 500×500 pixels [24]. The objective is to retrieve from the database stamps with visual color resemblance to a query stamp.

To apply the framework to this database, we need to devise a matching distance function that measures the distance between two images, 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 query image. These issues are addressed in the sections to follow.

4.1. A matching distance function

To apply the vantage-object framework to our set of stamp images, we need to provide a distance function giving the similarity between two images. Our approach is as follows. We split the HSV information of each image into three separate histograms: one each for hue, saturation, and value. Each histogram consists of a fixed number b of bins — 16 in our implementation, but this number was chosen somewhat arbitrarily and does not appear crucial.

We measure the similarity between two histograms H and H' by their difference, given by

$$d(H,H') = \sum_{i=1}^{b} |H_i - H'_i|,$$

where H_i is the fraction (with $0 \leqslant H_i \leqslant 1$) of pixels of H in bin i. Intuitively, this measures the amount of resemblance between the two histograms: if $d(H,H') = 0$, the histograms are identical. It is easily verified that this distance measure satisfies the properties listed in Section 2: (one-way) identity, positiveness, and the triangle inequality. The distance between two histograms can clearly be computed in $O(b)$ time, where b is the number of bins in each histogram. (As an alternative we also considered using the standard histogram-intersection approach described by Swain and Ballard [25], but their method was observed not to perform as well for our particular application.)

This distance function measures the direct resemblance of two histograms. In terms of, for example, hue information, two histograms will achieve a small distance if and only if the colors in the corresponding images are alike. In some applications however it could be preferable to consider the distribution of values over the histogram rather than the actual histogram. In terms of hue information, this means that we would consider two images similar if the hue of one is similar to (but shifted with respect to) that of the other; in terms of value information, two images would thus be considered similar if one is a darker or lighter copy of the other. The corresponding distance measure is given by

$$d(H,H') = \min_{0 \leqslant j < b} \sum_{i=1}^{b} |H_i - H'_{(i+j) \bmod b}|.$$

This remains a subject for future research.

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 of an object from each vantage object will be approximately similar, and little information is gained by adding the extra vantage objects.

For example, when dealing with hue information it makes sense for each vantage object to contain dominant colors that differ from those found in the other vantage objects. To obtain such a set of objects we picked from the database an image with well-defined hue, saturation, and value content, and modified these properties one by one to obtain a number of vantage objects measuring distinct image qualities. For the hue content we shifted the hue component of the original image by multiples of $90°$, and similarly varied the saturation and value contents in four discrete steps. This process resulted in the series of vantage objects shown at the top of Fig. 4. Note that the first four of these vantage objects impose order only on the hue content of the images, whereas the next four vantage objects are used strictly for their saturation content, and the last four only measure value. We thus end up with a twelve-dimensional vantage space.

Since the HSV components of an image are independent we would combine these 12 vantage objects into four composite objects, each of which measuring hue as well as saturation and value. The current approach however has two advantages: the role of each vantage object is intuitively clearer, and more important, it provides the option of weighting the HSV components differently. For example, the user might specify that he considers hue the most important property and wishes to retrieve only images with hue similar to that of the query images, whereas saturation and value are of subordinate importance.

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

[9–11]. Of particular interest is the approximate-nearest-neighbor algorithm due to Arya et al. [8], 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 $\varepsilon > 0$, a point $p \in P$ is a $(1+\varepsilon)$-*approximation* to the nearest neighbor of q if, for all $p' \in P$, we have $d(p,q) \leqslant (1+\varepsilon) \cdot d(p',q)$. A set of n points in $\mathbb{R}^d$ can be preprocessed in $O(n \log n)$ time and $O(n)$ storage, such that given a query point $q \in \mathbb{R}^d$, a constant $\varepsilon > 0$, and a constant integer $k \geqslant 1$, we can compute $(1+\varepsilon)$-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 $\varepsilon = 0$.

According to Theorem 3.2, a single query for m vantage images among n images thus takes $O(mT + k \log n)$ time, where k is the number of images returned, and T is the time required to compare any two given images. For our distance function this is $O(mb + k \log n) = O(m + k \log n)$, where b is the (constant) number of bins in each histogram.

As a possible postprocessing step, we may explicitly measure the similarity between the query image and each of the k images returned by the algorithm; this takes additional $O(kb) = O(k)$ 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 is to demonstrate the usefulness of the vantage-object structure by itself, even without further postprocessing of the query results.

4.4. Experimental results

We implemented the algorithm described in the previous sections, using an available implementation of the search algorithm due to Arya et al. [8], on a 400 MHz Pentium II-based workstation.

Three example queries and their results are shown in Fig. 4. The first query shows a single stamp from a series of nearly identical stamps with different nominations, whereas the second query returns very different stamps that are nevertheless similar in appearance. The third query consists of part of an envelope (rotated 90° clockwise) with three brightly colored stamps and some black postage marks on it, and nicely returns only similar images. The fact that only parts of envelopes (and no single stamps) are returned can be accounted for by the dominant presence of the color white in these images.

Computing a single histogram from a JPEG image takes 239.7 ms on average, whereas comparing two histograms takes approximately 27.1 ms. For a query image we first compute its three HSV histograms in approximately 0.72 s. Determining the corresponding vantage-space point is done by comparing these histograms against the 12 vantage objects in 325 ms. Finally, the nearest-neighbor query takes in all practical cases less than 15 ms. Thus the total time required for a query amounts to approximately 1 s. For comparison purposes: the brute-force approach of directly comparing the HSV histograms of the query image against all 5246 database images (assuming their histograms have been precomputed) takes well over seven minutes.

5. Shape matching: hieroglyphics

To demonstrate that our approach is applicable to many types of matching problems, we also considered the problem of image retrieval based not on color information but on shape. For this we used a collection of 6845 hieroglyphics with a total of 72,816 polylines [26]. Given a query hieroglyphic, the objective is to be able to retrieve similarly shaped hieroglyphics from the collection, that is, hieroglyphics containing polylines that resemble those of the query hieroglyphic. The various details that need to be filled in are considered in the following sections.

5.1. A matching distance function

Arkin et al. [27] describe a matching metric for polygons that is invariant under translation, rotation, and scaling, is defined for both convex and nonconvex 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

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

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

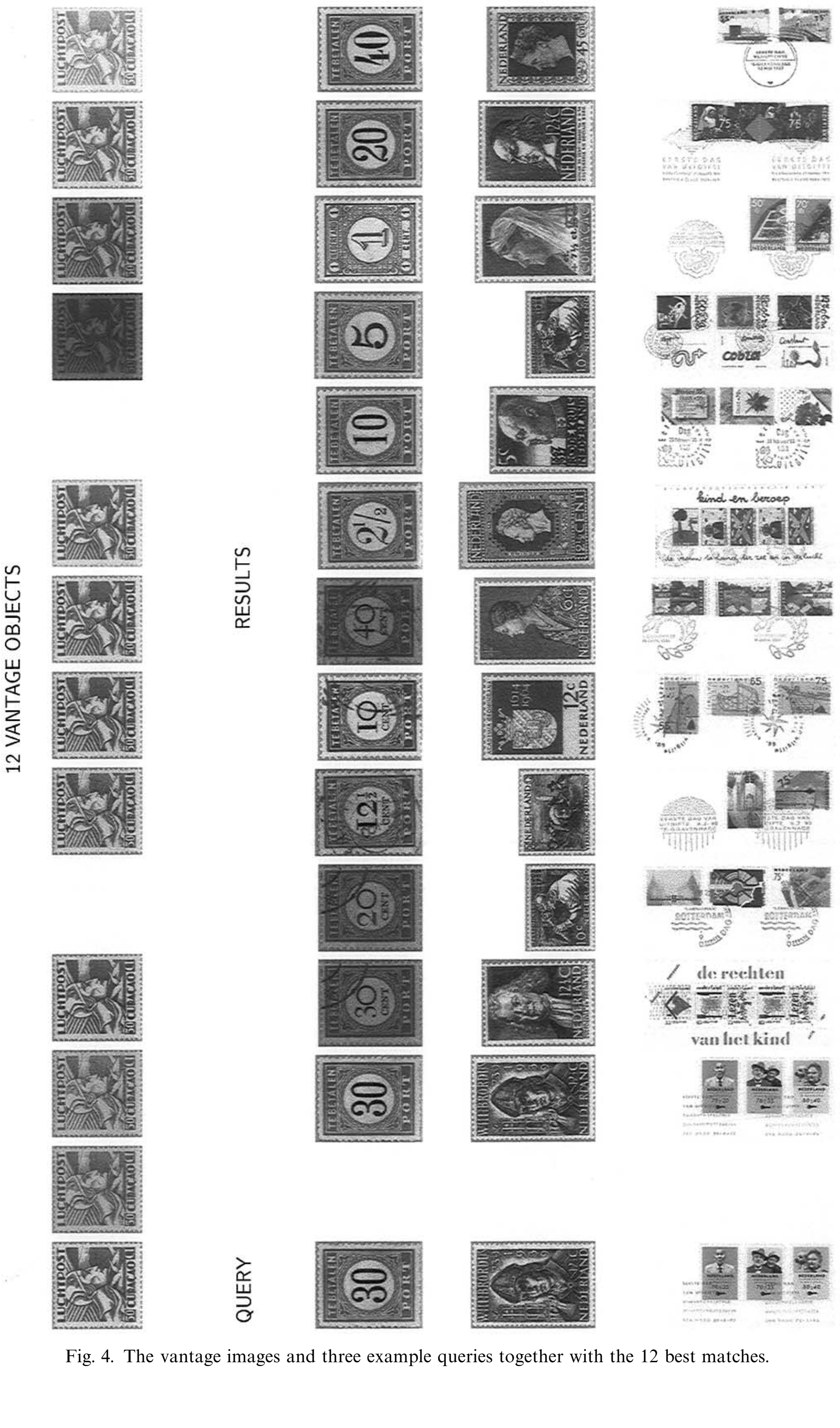

Fig. 4. The vantage images and three example queries together with the 12 best matches.

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 [28].

Let P and Q be two polylines with lengths l_P and l_Q, respectively. In consideration of the above issues we define the distance $d(P,Q)$ of Q from P as follows. First compute their turning functions $\Theta_P(s)$ and $\Theta_Q(s)$. Notice that (since we are dealing with polylines rather than polygones) these turning functions are not periodic. Moreover, their domains $\mathrm{Dom}(P) = [0, l_P]$ and $\mathrm{Dom}(Q) = [0, l_Q]$ — that is, the range over which s varies — most likely differ. Assuming without loss of generality that $l_P \geqslant l_Q$, we define the distance of Θ_Q from Θ_P as

$$d(\Theta_P,\Theta_Q) = \min_{0\leqslant t\leqslant l_P - l_Q,\, 0\leqslant \alpha < 2\pi} \int_0^{l_Q} |\Theta_P(s+t) - \Theta_Q(s) + \alpha|\, \mathrm{d}s.$$

In words: we shift Θ_Q (the shorter of the turning functions) along Θ_P while keeping track of the minimum area between the functions; only the portion of Θ_P that falls within the domain of Θ_Q is considered in this. In the case that $l_P < l_Q$, we define $d(\Theta_P,\Theta_Q) = d(\Theta_Q,\Theta_P)$ — note that this implies that our distance function is symmetric. It is easily verified that this distance function implements partial polyline matching, and $d(P,Q) = 0$ if and only if P occurs in its entirety in Q or vice versa.

This distance function however does not satisfy the triangle inequality. Consider the polylines shown in Fig. 5: since an exact copy of P_1 occurs in P_2 we have $d(P_1,P_2) = 0$, and similarly $d(P_2,P_3) = d(P_3,P_2) = 0$. P_1 and P_3 however differ considerably and therefore $d(P_1,P_3) > 0$, which violates the triangle inequality $d(P_1,P_3) \leqslant d(P_1,P_2) + d(P_2,P_3)$. Since the proof Lemma 3.1 relies on this property to hold our current approach is not guaranteed to be complete — that is, it may miss

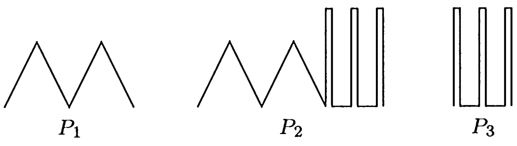

Fig. 5. Three polylines for which the distance function violates the triangle inequality.

possible matches to a query. Nevertheless, the experimental results to be presented in Section 5.4 indicates that in practice this violation does not render our approach useless. It is interesting to speculate on the reasons for this. A possible factor is that the triangle inequality will usually hold for objects that occur in practice – the example of Fig. 5 is highly artificial. Moreover, the inequality need only hold for objects similar to the query object: the proof to Lemma 3.1 considers objects A_i such that $d(A_?, A_i) \leqslant \varepsilon$ and $d(A_i, A_?) \leqslant \varepsilon$. We consider this a topic for future research.

Omitting details about the implementation of the above distance function, we state that our matching algorithm runs in identical $T = O(c_1c_2 \log c_1c_2)$ 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* number of segments is much lower: about 14.2) By Theorem 3.2, this yields a bound on retrieving the k nearest neighbors of a query polyline among n polylines of $O(mc^2 \log c + k \log n) = O(m + k \log n)$, where m is the number of vantage polylines, and c the (constant) maximum number of segments to a polyline. The (intentionally unimplemented) postprocessing step of explicitly comparing the query polyline to each of the k polylines returned by the algorithm, as discussed in Section 4.3, takes additional $O(kc^2 \log c) = O(k)$ time and therefore does not influence this bound.

5.2. Choosing vantage objects

As discussed in Section 4.2, the vantage objects should measure different properties of the database 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.

An additional concern lies with the complexity of the vantage objects. Even though our approach was devised to require only few runtime distance calculations, still performance could be impaired if we pick exceptionally complex vantage objects for which the distance function is difficult to compute.

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.

5.3. Matching entire hieroglyphics

So far, our algorithm description focused on matching single polylines. The hieroglyphics in our database however consists of several — on average about 14 — 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. Therefore we sort the hieroglyphics matched using the following simple heuristics. The more polylines of a given hieroglyphic match closely — that is, within the distance threshold — 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 any of the matched polylines 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. Note that this is functionally equivalent to using an axis-parallel range search (with the query range centered at the query point) instead of a nearest-neighbor query.

5.4. Experimental results

We implemented the algorithm described in this section on the same 400 MHz Pentium II-based workstation. As in the application to stamps we use the approximate-nearest-neighbor algorithm due to Arya et al. for determining the nearest vantage-space neighbors of a query point.

Some typical queries for $m = 6$ vantage points, along with the hieroglyphics returned and the vantage polylines used, are shown in Fig. 6. Note that the most complicated of the vantage polylines consists of 39 segments, thus ensuring efficient distance calculations (see Section 5.2). For these queries, we set the minimum number of segments that a polyline should consist of to three — in other words, 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 copy of the shape — more or less, as the hieroglyphics have been digitized by hand. 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. From experiments we learned that for our data set the distance measure takes 23.3 ms on average to compute. It follows that comparing all 72,816 polylines against a single polyline takes approximately 28 min, and consequently the preprocessing step of computing the vantage-space points for $m = 6$ vantage objects takes approximately 2 h and 48 min. When we want to add polylines to the database, this computation need only be performed for the polylines to be added. Finally, the time required for building the nearest-neighbor data structure depends on the number m of vantage objects but is, for example, only slightly higher than three seconds for $m = 10$. Moreover, this step has to be performed only when the image database changes.

The query time in turn is low due to the intensive preprocessing steps. For a query we first have to compute its vantage-space coordinates by computing its distance from the vantage objects, which in our application takes 6×23.3 ms ≈ 140 ms. Next, we need to determine the nearest neighbors of the corresponding point. The time required for this depends on the distance threshold but does not exceed 10 ms for all practical values of ε. Thus the total time for a query is approximately 150 ms.

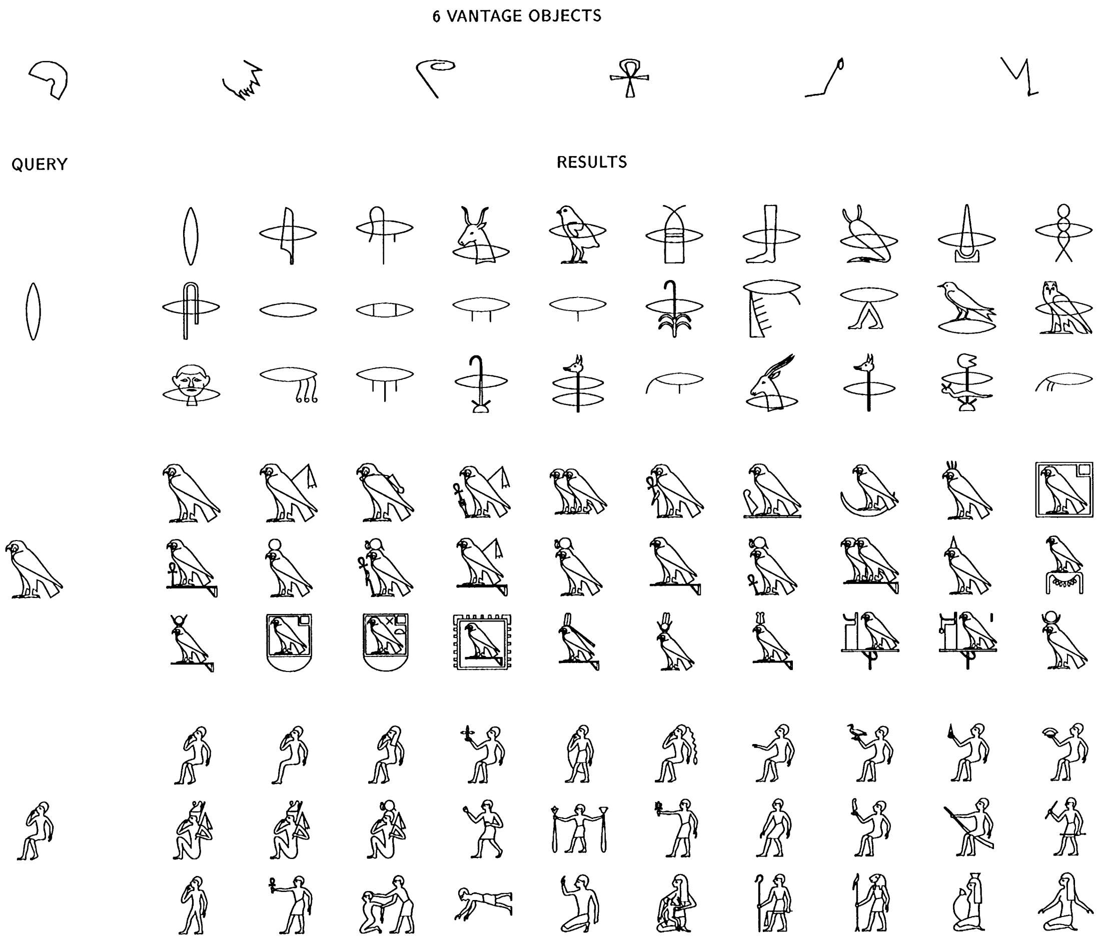

Fig. 6. The vantage objects used to define the vantage space, and three example queries along with their 30 best matches.

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 almost half an hour. To match an entire query hieroglyphic this time should be multiplied by the number of polylines the hieroglyphic contains.

Another issue is the number of vantage objects. Table 1 summarizes the number k of results returned 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 number of results quickly grows if we use 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 for a larger number of vantage points.

Table 1
Comparison for different numbers of vantage points and distance thresholds

	$\varepsilon = 0.05$	$\varepsilon = 0.10$	$\varepsilon = 0.25$	$\varepsilon = 0.50$	$\varepsilon = 1.00$
m	k	k	k	k	k
1	3312	6387	15010	24944	> 30000
2	76.8	244	1261	3833	11692
3	8.8	54.2	375	1182	4742
4	3.0	18.4	195	817	3021
6	1.2	8.2	71.8	483	1911
8	1.0	3.2	46.8	215	1338
10	1.0	1.2	34.8	164	1073

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 histogram- and shape-based image similarity. An important advantage of our structure is that a query requires only a small constant number of distance calculations between database objects; other approaches usually perform at least $O(\log n)$ such calculations. This becomes especially beneficial if the similarity measure is expensive to compute.

Although the viability of our approach was demonstrated only for two specific cases — retrieving raster images of stamps, and vector images depicting hieroglyphics — its potential is still more general. In fact, our indexing structure applies to any set of images for which a similarity distance function can be defined. Since almost all distance calculation are performed during an offline preprocessing step, another 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 10 or less — of distance calculations. If sufficient preprocessing time is available, one can therefore use an elaborate (and possibly expensive) distance function with desirable properties such as robustness against noise, blurrings, cracks, and deformations [29].

Future work includes implementing a number of standard approaches against which to compare our implementation, as our current experimental results provide comparisons only against a straightforward brute-force approach.

References

[1] V.E. Ogle, M. Stonebraker, Chabot: retrieval from a relational database of images, IEEE Comput. 28 (1995) 40–48.

[2] A. Pentland, R.W. Picard, S. Sclaroff, Photobook: tools for content-based manipulation of image databases, Proceedings of the SPIE: Storage and Retrieval for Image and Video Databases II, Vol. 2185, 1994, pp. 34–47.

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

[4] P.M. Kelly, T.M. Cannon, D.R. Hush, Query by image example: the CANDID approach, Proceedings of the SPIE: Storage and Retrieval for Image and Video Databases III, Vol. 2420, 1995, pp. 238–248.

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

[6] Y. Lamdan, J.T. Schwartz, H.J. Wolfson, Geometric hashing: a general and efficient model-based recognition scheme, Proceedings of the International Conference on Computer Vision, 1988, pp. 238–249.

[7] H.J. Wolfson, Model-based object recognition by geometric hashing, Proceedings of the First European Conference on Computer Vision, 1990, pp. 526–536.

[8] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching, Proceedings of the Fifth ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 573–582. An implementation is available from `http://www.cs.umd.edu/~mount/ANN`.

[9] J.L. Bentley, *K*-d trees for semidynamic point sets, Proceedings of the Sixth Annual ACM Symposium on Computer Geometry 1990, pp. 187–197.

[10] K.L. Clarkson, Nearest neighbor queries in metric spaces, Proceedings of the 29th Annual ACM Symposium on Theory and Computation, 1997, pp. 609–617.

[11] J. Kleinberg, Two algorithms for nearest-neighbor search in high dimension, Proceedings of the 29th Annual ACM Symposium on Theory and Computation 1997, pp. 599–608.

[12] W.A. Burkhard, R.M. Keller, Some approaches to best-match file searching, Commun. ACM 16 (4) (1973) 230–236.

[13] P.N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, Proceedings of the Fourth ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 311–321.

[14] J.K. Uhlmann, Satisfying general proximity/similarity queries with metric trees, Inform. Process. Lett. 40 (1991) 175–179.

[15] S. Berchtold, D.A. Keim, H.-P. Kriegel, The X-tree: An index structure for higher dimensional data, Proceedings of the 22th VLDB Conference, 1996, pp. 28–39.

[16] D.A. White, R. Jain, Similarity indexing with the SS-tree, Proceedings of the 12th IEEE International Conference on Data Engineering, 1996, pp. 516–523.

[17] N. Katayama, S. Satoh, The SR-tree: an index structure for high-dimensional nearest neighbor queries, SIGMOD '97, 1997, pp. 369–380.

[18] K.I. Lin, H.V. Jagdish, C. Faloutsos, The TV-tree: an index structure for higher dimensional data, VLDB J. 4 (1994) 517–542.

[19] J. Matoušek, Efficient partition trees, Discrete Comput. Geom. 8 (1992) 315–334.

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

[21] B. Chazelle, E. Welzl, Quasi-optimal range searching in spaces of finite VC-dimension, Discrete Comput. Geom. 4 (1989) 467–489.

[22] O. Schwarzkopf, J. Vleugels, Range searching in low-density environments, Inform. Process. Lett. 60 (1996) 121–127.

[23] J. Vleugels, On fatness and fitness—realistic input models for geometric algorithms, Ph.D. Thesis, Department of Computer Science, University of Utrecht, Utrecht, The Netherlands, 1997.

[24] Software Generation, Collect-A-ROM: Postzegels Nederland en overzee, 1996.

[25] M.J. Swain, D.H. Ballard, Color indexing, Int. J. Comput. Vision 7 (1991) 11–32.

[26] 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.
[27] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, J.S.B. Mitchell, An efficiently computable metric for comparing polygonal shapes, IEEE Trans. Pattern Anal. Mach. Intell. 13 (3) (1991) 209–216.
[28] S.D. Cohen, L.J. Guibas, Partial matching of planar polylines under similarity transformations, Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms, January 1997, pp. 777–786.
[29] M. Hagedoorn, R.C. Veltkamp, A robust affine invariant metric on boundary patterns, Int. J. Pattern Recognition Artif. Intell. 13 (1999) 1151–1164.

About the Author—JULES VLEUGELS received his Ph.D. degree in Computer Science from Utrecht University in 1998. He has worked on motion planning and computational geometry, focusing on realistic input models for geometric algorithms. His current research interests as a postdoc at Utrecht University include data structures for content-based image retrieval.

About the Author—REMCO C. VELTKAMP is Assistant Professor at Utrecht University. He has worked on surface reconstruction, approximation, spline modeling, and constraint-based 00 graphics. His current research focuses on shape algorithmics, the application of Computational Geometry in computer vision and shape processing, and the application of shape matching in content-based image retrieval. Aspects like algorithmic design and experimental verification play an important role in his work.